ADOBE® FLASH® LITE™ 2.x and 3.x Adobe® ActionScript® Language Reference

© 2010 Adobe Systems Incorporated. All rights reserved.

Adobe® Flash® Lite® 2.x and 3.x ActionScript® Language Reference

This Language Reference is licensed for use under the terms of the Creative Commons Attribution Non-Commercial 3.0 License. This License allows users to copy, distribute, and transmit the user guide for noncommercial purposes only so long as (1) proper attribution to Adobe is given as the owner of the guide; and (2) any reuse or distribution of the guide contains a notice that use of the user guide is governed by these terms. The best way to provide notice is to include the following link. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/

Adobe, the Adobe logo, ActionScript, Flash, and Flash Lite are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Windows, Windows NT, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Last updated 12/20/2010

Contents

Chapter 1: ActionScript language elements

Compiler directives 1
Constants 4
Global functions 8
Global properties 55
Operators 71
Statements 122
fscommand2 commands 160

Chapter 2: ActionScript classes

arguments 178
Array 179
BitmapData (flash.display.BitmapData) 197
Boolean 218
Button 220
capabilities (System.capabilities) 241
Color 259
ColorTransform (flash.geom.ColorTransform) 263
Date 276
Error 302
ExtendedKey 306
Function 310
Key 313
LoadVars 325
LocalConnection 335
Math 348
Matrix (flash.geom.Matrix) 362
Mouse 382
MovieClip 388
MovieClipLoader 459
NetConnection 472
NetStream 474
Number 488
Object 493
Point (flash.geom.Point) 508
Rectangle (flash.geom.Rectangle) 517
Security (System.security) 538
Selection 542
SharedObject 549
Sound 560
Stage 579
String 584

System 597
TextField 599
TextFormat 639
Transform (flash.geom.Transform) 652
Video 659
XML 665
XMLNode 683
XMLSocket 699

Chapter 3: Deprecated ActionScript
Deprecated Function summary 707
Deprecated Property summary 708
Deprecated Operator summary 709

Chapter 4: Unsupported ActionScript
Unsupported Classes 710
Unsupported Methods 710
Unsupported Properties 710
Unsupported Global Functions 710
Unsupported Event Handlers 711
Unsupported fscommands 711

Index 712

Chapter 1: ActionScript language elements

This section provides syntax, usage information, and code samples for global functions and properties (those elements that do not belong to an ActionScript class); compiler directives; and for the constants, operators, statements, and keywords used in ActionScript and defined in the ECMAScript (ECMA-262) edition 4 draft language specification.

Compiler directives

This section contains the directives to include in your ActionScript file to direct the compiler to preprocess certain instructions.

Compiler directives summary

Directive	Description
#endinitclip	Compiler directive; indicates the end of a block of initialization actions.
#include	Compiler directive; includes the contents of the specified file, as if the commands in the file are part of the calling script.
#initclip	Compiler directive; indicates the beginning of a block of initialization actions.

#endinitclip directive

```
#endinitclip
```

Compiler directive; indicates the end of a block of initialization actions.

Availability

Flash Lite™ 2.0

Example

```
#initclip
...initialization actions go here...
#endinitclip
```

#include directive

```
#include "[path]filename.as"
```

Note: *Do not place a semicolon (;) at the end of the line that contains the #include statement.*

Compiler directive; includes the contents of the specified file, as if the commands in the file are part of the calling script. The `#include` directive is invoked at compile time. Therefore, if you make any changes to an external file, you must save the file and recompile any FLA files that use it.

If you use the Check Syntax button for a script that contains `#include` statements, the syntax of the included files is also checked.

You can use `#include` in FLA files and in external script files, but not in ActionScript 2.0 class files.

You can specify no path, a relative path, or an absolute path for the file to be included. If you don't specify a path, the AS file must be in one of the following locations:

- The same directory as the FLA file. The same directory as the script containing the `#include` statement
- The global Include directory, which is one of the following:

  Windows® 2000 or Windows XP: C:\Documents and Settings*user* \Local

  Settings\ Application Data\Adobe\Flash 10*language*\Configuration\Include

  Windows Vista®: C:\Users*user* \Local Settings\ Application Data\Adobe\Flash 8*language*\Configuration\Include

  Macintosh® OS X: Hard Drive/Users/Library/Application Support/Adobe/Flash 10/*language*/Configuration/Include
- The *Flash program**language*\First Run\Include directory; if you save a file here, it is copied to the global Include directory the next time you start Flash®.

To specify a relative path for the AS file, use a single dot (.) to indicate the current directory, two dots (. .) to indicate a parent directory, and forward slashes (/) to indicate subdirectories. See the following example section.

To specify an absolute path for the AS file, use the format supported by your platform (Macintosh or Windows). See the following example section. (This usage is not recommended because it requires the directory structure to be the same on any computer that you use to compile the script.)

Note: *If you place files in the First Run/Include directory or in the global Include directory, back up these files. If you ever need to uninstall and reinstall Flash, these directories might be deleted and overwritten.*

Availability

Flash Lite 2.0

Parameters

`[path]filename.as` - *filename.as* The filename and optional path for the script to add to the Actions panel or to the current script; *.as* is the recommended filename extension.

Example

The following examples show various ways of specifying a path for a file to be included in your script:

```
// Note that #include statements do not end with a semicolon (;)
// AS file is in same directory as FLA file or script
// or is in the global Include directory or the First Run/Include directory
#include "init_script.as"

// AS file is in a subdirectory of one of the above directories
// The subdirectory is named "FLA_includes"
#include "FLA_includes/init_script.as"
// AS file is in a subdirectory of the script file directory
// The subdirectory is named "SCRIPT_includes"
#include "SCRIPT_includes/init_script.as"
// AS file is in a directory at the same level as one of the above directories
// AS file is in a directory at the same level as the directory
// that contains the script file
// The directory is named "ALL_includes"
#include "../ALL_includes/init_script.as"

// AS file is specified by an absolute path in Windows
// Note use of forward slashes, not backslashes
#include "C:/Flash_scripts/init_script.as"

// AS file is specified by an absolute path on Macintosh
#include "Mac HD:Flash_scripts:init_script.as"
```

#initclip directive

`#initclip` *`order`*

Note: *Do not place a semicolon (;) at the end of the line that contains the #initclip statement.*

Compiler directive; indicates the beginning of a block of initialization actions. When multiple clips are initialized at the same time, you can use the `order` parameter to specify which initialization occurs first. Initialization actions execute when a movie clip symbol is defined. If the movie clip is an exported symbol, the initialization actions execute before the actions on Frame 1 of the SWF file. Otherwise, they execute immediately before the frame actions of the frame that contains the first instance of the associated movie clip symbol.

Initialization actions execute only once when a SWF file plays; use them for one-time initializations, such as class definition and registration.

Availability
Flash Lite 2.0

Parameters
`order` - A non-negative integer that specifies the execution order of blocks of `#initclip` code. This is an optional parameter. You must specify the value by using an integer literal (only decimal—not hexadecimal—values are allowed), and not by using a variable. If you include multiple `#initclip` blocks in a single movie clip symbol, then the compiler uses the last `order` value specified in that movie clip symbol for all `#initclip` blocks in that symbol.

Example
In the following example, ActionScript is placed on Frame 1 inside a movie clip instance. A variables.txt text file is placed in the same directory.

```
#initclip

trace("initializing app");

var variables:LoadVars = new LoadVars();

variables.load("variables.txt");

variables.onLoad = function(success:Boolean) {

    trace("variables loaded:"+success);

    if (success) {
    for (i in variables) {
    trace("variables."+i+" = "+variables[i]);
    }
    }
};

#endinitclip
```

Constants

A constant is a variable used to represent a property whose value never changes. This section describes global constants that are available to every script.

Constants summary

Modifiers	Constant	Description
	false	A unique Boolean value that represents the opposite of `true`.
	Infinity	Specifies the IEEE-754 value representing positive infinity.
	-Infinity	Specifies the IEEE-754 value representing negative infinity.
	NaN	A predefined variable with the IEEE-754 value for NaN (not a number).
	newline	Inserts a carriage return character (`\r`) that generates a blank line in text output generated by your code.
	null	A special value that can be assigned to variables or returned by a function if no data was provided.
	true	A unique Boolean value that represents the opposite of `false`.
	undefined	A special value, usually used to indicate that a variable has not yet been assigned a value.

false constant

A unique Boolean value that represents the opposite of `true`.

When automatic data typing converts `false` to a number, it becomes 0; when it converts `false` to a string, it becomes `"false"`.

Availability
Flash Lite 1.1

Example
This example shows how automatic data typing converts `false` to a number and to a string:

```
var bool1:Boolean = Boolean(false);

// converts it to the number 0
trace(1 + bool1); // outputs 1

// converts it to a string
trace("String: " + bool1); // outputs String: false
```

Infinity constant

Specifies the IEEE-754 value representing positive infinity. The value of this constant is the same as `Number.POSITIVE_INFINITY`.

Availability
Flash Lite 2.0

See also
`POSITIVE_INFINITY (Number.POSITIVE_INFINITY property)`

-Infinity constant

Specifies the IEEE-754 value representing negative infinity. The value of this constant is the same as `Number.NEGATIVE_INFINITY`.

Availability
Flash Lite 2.0

See also
`NEGATIVE_INFINITY (Number.NEGATIVE_INFINITY property)`

NaN constant

A predefined variable with the IEEE-754 value for NaN (not a number). To determine if a number is NaN, use `isNaN()`.

Availability
Flash Lite 1.1

See also
`isNaN function`, `NaN (Number.NaN property)`

newline constant

Inserts a carriage return character (`\r`) that generates a blank line in text output generated by your code. Use `newline` to make space for information that is retrieved by a function or statement in your code.

Availability

Flash Lite 1.1

Example

The following example shows how `newline` displays output from the `trace()` statement on multiple lines.

```
var myName:String = "Lisa", myAge:Number = 30;
trace(myName+myAge);
trace("-----");
trace(myName+newline+myAge);
// output:
Lisa30
-----
Lisa
30
```

See also

`trace function`

null constant

A special value that can be assigned to variables or returned by a function if no data was provided. You can use `null` to represent values that are missing or that do not have a defined data type.

Availability

Flash Lite 1.1

Example

In a numeric context, `null` evaluates to 0. Equality tests can be performed with `null`. In this statement, a binary tree node has no left child, so the field for its left child could be set to `null`.

```
if (tree.left == null) {
    tree.left = new TreeNode();
}
```

true constant

A unique Boolean value that represents the opposite of `false`. When automatic data typing converts `true` to a number, it becomes 1; when it converts `true` to a string, it becomes `"true"`.

Availability

Flash Lite 1.1

Example

The following example shows the use of `true` in an `if` statement:

```
var shouldExecute:Boolean;
// ...
// code that sets shouldExecute to either true or false goes here
// shouldExecute is set to true for this example:

shouldExecute = true;

if (shouldExecute == true) {
    trace("your statements here");
}

// true is also implied, so the if statement could also be written:
// if (shouldExecute) {
// trace("your statements here");
// }
```

The following example shows how automatic data typing converts `true` to the number 1:

```
var myNum:Number;
myNum = 1 + true;
trace(myNum); // output: 2
```

See also

`false constant`, `Boolean`

undefined constant

A special value, usually used to indicate that a variable has not yet been assigned a value. A reference to an undefined value returns the special value `undefined`. The ActionScript code `typeof(undefined)` returns the string `"undefined"`. The only value of type `undefined` is `undefined`.

In files published for Flash Player 6 or earlier, the value of `String(undefined)` is "" (an empty string). In files published for Flash Player 7 or later, the value of `String(undefined)` is `"undefined"` (`undefined` is converted to a string).

In files published for Flash Player 6 or earlier, the value of `Number(undefined)` is `0`. In files published for Flash Player 7 or later, the value of `Number(undefined)` is `NaN`.

The value `undefined` is similar to the special value `null`. When `null` and `undefined` are compared with the equality (`==`) operator, they compare as equal. However, when `null` and `undefined` are compared with the strict equality (`===`) operator, they compare as not equal.

Availability

Flash Lite 1.1

Example

In the following example, the variable `x` has not been declared and therefore has the value `undefined`.

In the first section of code, the equality operator (`==`) compares the value of `x` to the value `undefined`, and the appropriate result is sent to the Output panel. In the first section of code, the equality operator (`==`) compares the value of `x` to the value `undefined`, and the appropriate result is sent to the log file.

In the second section of code, the equality (`==`) operator compares the values `null` and `undefined`.

```
// x has not been declared
trace("The value of x is "+x);

if (x == undefined) {
    trace("x is undefined");
} else {
    trace("x is not undefined");
}

trace("typeof (x) is "+typeof (x));

if (null == undefined) {
    trace("null and undefined are equal");
} else {
    trace("null and undefined are not equal");
}
```

The following result is displayed in the Output panel.

```
The value of x is undefined
x is undefined
typeof (x) is undefined
null and undefined are equal
```

Global functions

This section contains a set of built-in functions that are available in any part of a SWF file where ActionScript is used. These global functions cover a wide variety of common programming tasks such as working with data types (`Boolean()`, `int()`, and so on), producing debugging information (`trace()`), and communicating with Flash Player or the browser (`fscommand()`).

Global functions summary

Modifiers	Signature	Description
	Array `([numElements], [elementN]) : Array`	Creates a new, empty array or converts specified elements to an array.
	Boolean `(expression:Object) : Boolean`	Converts the parameter ***`expression`*** to a Boolean value and returns `true` or `false`.
	call `(frame:Object)`	**Deprecated** since Flash Player 5. This action was deprecated in favor of the `function` statement. Executes the script in the called frame without moving the playhead to that frame.
	chr `(number:Number) : String`	**Deprecated** since Flash Player 5. This function was deprecated in favor of `String.fromCharCode()`. Converts ASCII code numbers to characters.
	clearInterval `(intervalID:Number)`	Cancels an interval created by a call to `setInterval()`.
	duplicateMovieClip `(target:Object, newname:String, depth:Number)`	Creates an instance of a movie clip while the SWF file is playing.

Modifiers	Signature	Description
	escape (expression:String) : String	Converts the parameter to a string and encodes it in a URL-encoded format, where all nonalphanumeric characters are replaced with % hexadecimal sequences.
	eval (expression:Object) : Object	Accesses variables, properties, objects, or movie clips by name.
	fscommand (command:String, parameters:String)	Lets a SWF file communicate with the Flash Lite player or the environment for a mobile device (such as an operating system).
	fscommand2 (command:String, parameters:String)	Lets the SWF file communicate with the Flash Lite player or a host application on a mobile device.
	getProperty (my_mc:Object, property:Object) : Object	**Deprecated** since Flash Player 5. This function was deprecated in favor of the dot syntax, which was introduced in Flash Player 5. Returns the value of the specified property for the movie clip ***my_mc***.
	getTimer () : Number	Returns the number of milliseconds that have elapsed since the SWF file started playing.
	getURL (url:String, [window:String], [method:String])	Loads a document from a specific URL into a window or passes variables to another application at a defined URL.
	getVersion () : String	Returns a string containing Flash Player version and platform information.
	gotoAndPlay ([scene:String], frame:Object)	Sends the playhead to the specified frame in a scene and plays from that frame.
	gotoAndStop ([scene:String], frame:Object)	Sends the playhead to the specified frame in a scene and stops it.
	ifFrameLoaded ([scene:String], frame:Object, statement(s):Object)	**Deprecated** since Flash Player 5. This function has been deprecated. Adobe recommends that you use the `MovieClip._framesloaded` property. Checks whether the contents of a specific frame are available locally.
	int (value:Number) : Number	**Deprecated** since Flash Player 5. This function was deprecated in favor of `Math.round()`. Converts a decimal number to an integer value by truncating the decimal value.
	isFinite (expression:Object) : Boolean	Evaluates `expression` and returns `true` if it is a finite number or `false` if it is infinity or negative infinity.
	isNaN (expression:Object) : Boolean	Evaluates the parameter and returns `true` if the value is `NaN` (not a number).
	length (expression:String, variable:Object) : Number	**Deprecated** since Flash Player 5. This function, along with all the string functions, has been deprecated. Adobe recommends that you use the methods of the String class and the `String.length` property to perform the same operations. Returns the length of the specified string or variable.
	loadMovie (url:String, target:Object, [method:String])	Loads a SWF or JPEG file into Flash Player while the original SWF file plays.

Modifiers	Signature	Description
	loadMovieNum (url:String, level:Number, [method:String])	Loads a SWF or JPEG file into a level in Flash Player while the originally loaded SWF file plays.
	loadVariables (url:String, target:Object, [method:String])	Reads data from an external file, such as a text file or text generated by ColdFusion, a CGI script, Active Server Pages (ASP), PHP, or Perl script, and sets the values for variables in a target movie clip.
	loadVariablesNum (url:String, level:Number, [method:String])	Reads data from an external file, such as a text file or text generated by a ColdFusion, CGI script, ASP, PHP, or Perl script, and sets the values for variables in a Flash Player level.
	mbchr (number:Number)	**Deprecated** since Flash Player 5. This function was deprecated in favor of the `String.fromCharCode()` method. Converts an ASCII code number to a multibyte character.
	mblength (string:String) : Number	**Deprecated** since Flash Player 5. This function was deprecated in favor of the `String.length` property. Returns the length of the multibyte character string.
	mbord (character:String) : Number	**Deprecated** since Flash Player 5. This function was deprecated in favor of `String.charCodeAt()` method. Converts the specified character to a multibyte number.
	mbsubstring (value:String, index:Number, count:Number) : String	**Deprecated** since Flash Player 5. This function was deprecated in favor of `String.substr()` method. Extracts a new multibyte character string from a multibyte character string.
	nextFrame ()	Sends the playhead to the next frame.
	nextScene ()	Sends the playhead to Frame 1 of the next scene.
	Number (expression:Object) : Number	Converts the parameter `expression` to a number.
	Object ([value:Object]) : Object	Creates a new empty object or converts the specified number, string, or Boolean value to an object.
	on (mouseEvent:Object)	Specifies the mouse event or keypress that triggers an action.
	onClipEvent (movieEvent:Object)	Triggers actions defined for a specific instance of a movie clip.
	ord (character:String) : Number	**Deprecated** since Flash Player 5. This function was deprecated in favor of the methods and properties of the String class. Converts characters to ASCII code numbers.
	parseFloat (string:String) : Number	Converts a string to a floating-point number.
	parseInt (expression:String, [radix:Number]) : Number	Converts a string to an integer.
	play ()	Moves the playhead forward in the Timeline.
	prevFrame ()	Sends the playhead to the previous frame.
	prevScene ()	Sends the playhead to Frame 1 of the previous scene.

Modifiers	Signature	Description
	random (value:Number) : Number	**Deprecated** since Flash Player 5. This function was deprecated in favor of Math.random(). Returns a random integer between 0 and one less than the integer specified in the value parameter.
	removeMovieClip (target:Object)	Deletes the specified movie clip.
	setInterval (functionName:Object, interval:Number, [param:Object], objectName:Object, methodName:String) : Number	Calls a function or a method or an object at periodic intervals while a SWF file plays.
	setProperty (target:Object, property:Object, expression:Object)	Changes a property value of a movie clip as the movie clip plays.
	startDrag (target:Object, [lock:Boolean], [left,top,right,bottom:Number])	Makes the target movie clip draggable while the movie plays.
	stop ()	Stops the SWF file that is currently playing.
	stopAllSounds ()	Stops all sounds currently playing in a SWF file without stopping the playhead.
	stopDrag ()	Stops the current drag operation.
	String (expression:Object) : String	Returns a string representation of the specified parameter.
	substring (string:String, index:Number, count:Number) : String	**Deprecated** since Flash Player 5. This function was deprecated in favor of String.substr(). Extracts part of a string.
	targetPath (targetObject:Object) : String	Returns a string containing the target path of movieClipObject.
	tellTarget (target:String, statement(s):Object)	**Deprecated** since Flash Player 5. Adobe recommends that you use dot (.) notation and the with statement. Applies the instructions specified in the statements parameter to the Timeline specified in the target parameter.
	toggleHighQuality ()	**Deprecated** since Flash Player 5. This function was deprecated in favor of _quality. Turns anti-aliasing on and off in Flash Player.
	trace (expression:Object)	Evaluates the expression and outputs the result.
	unescape (string:String) : String	Evaluates the parameter *x* as a string, decodes the string from URL-encoded format (converting all hexadecimal sequences to ASCII characters), and returns the string.
	unloadMovie (target)	Removes a movie clip that was loaded by means of loadMovie() from Flash Player.
	unloadMovieNum (level:Number)	Removes a SWF or image that was loaded by means of loadMovieNum() from Flash Player.

Array function

`Array(): Array Array(numElements:Number): Array Array( [element0:Object [, element1, element2, ...elementN] ]) : Array`

Creates a new array of length zero or more, or an array populated by a list of specified elements, possibly of different data types.

Lets you create one of the following:

- an empty array
- an array with a specific length but whose elements have undefined values
- an array whose elements have specific values.

Using this function is similar to creating an array with the Array constructor (see "Constructor for the Array class").

You can pass a number (`numElements`) or a list of elements comprising one or more different types (`element0, element1, ..., elementN`).

Parameters that can accept more than one data type are listed as in the signature as type `Object`.

Availability
Flash Lite 2.0

Parameters
`numElements` [optional] - A positive integer that specifies the number of elements in the array. You can specify either numElements or the list of elements, not both.

`elementN` [optional] - one or more parameters, *element0, element1, ... , elementN*, the values of which can be of any type. Parameters that can accept more than one data type are listed as type `Object`. You can specify either `numElements` or the list of elements, not both.

Returns
`Array` - An array.

Example
```
var myArray:Array = Array();
myArray.push(12);
trace(myArray); //traces 12
myArray[4] = 7;
trace(myArray); //traces 12,undefined,undefined,undefined,7
```

Usage 2: The following example creates an array of length 4 but with no elements defined:

```
var myArray:Array = Array(4);
trace(myArray.length); // traces 4
trace(myArray); // traces undefined,undefined,undefined,undefined
```

Usage 3: The following example creates an array with three defined elements:

```
var myArray:Array = Array("firstElement", "secondElement", "thirdElement");
trace (myArray); // traces firstElement,secondElement,thirdElement
```

Note: *Unlike the Array class constructor, the Array() function does not use the keyword new.*

See also

Array

Boolean function

```
Boolean(expression:Object) : Boolean
```

Converts the parameter *expression* to a Boolean value and returns a value as described in the following list:

- If *expression* is a Boolean value, the return value is *expression*.
- If *expression* is a number, the return value is `true` if the number is not zero; otherwise the return value is `false`.

If *expression* is a string, the return value is as follows:

- In files published for Flash Player 6 or earlier, the string is first converted to a number; the value is `true` if the number is not zero, `false` otherwise.
- In files published for Flash Player 7 or later, the result is `true` if the string has a length greater than zero; the value is `false` for an empty string.

If *expression* is a string, the result is `true` if the string has a length greater than zero; the value is `false` for an empty string.

- If *expression* is `undefined` or `NaN` (not a number), the return value is `false`.
- If *expression* is a movie clip or an object, the return value is `true`.

Note: *Unlike the Boolean class constructor, the Boolean() function does not use the keyword new. Moreover, the Boolean class constructor initializes a Boolean object to false if no parameter is specified, while the Boolean() function returns undefined if no parameter is specified.*

Availability

Flash Lite 2.0

Parameters

`expression:Object` - An expression to convert to a Boolean value.

Returns

`Boolean` - A Boolean value.

Example

```
trace(Boolean(-1)); // output: true
trace(Boolean(0)); // output: false
trace(Boolean(1)); // output: true


trace(Boolean(true)); // output: true
trace(Boolean(false)); // output: false


trace(Boolean("true")); // output: true
trace(Boolean("false")); // output: true

trace(Boolean("Craiggers")); // output: true
trace(Boolean("")); // output: false
```

If files are published for Flash Player 6 and earlier, the results differ for three of the preceding examples:

```
trace(Boolean("true")); // output: false
trace(Boolean("false")); // output: false
trace(Boolean("Craiggers")); // output: false
```

This example shows a significant difference between use of the `Boolean()` function and the Boolean class. The `Boolean()` function creates a Boolean value, and the Boolean class creates a Boolean object. Boolean values are compared by value, and Boolean objects are compared by reference.

```
// Variables representing Boolean values are compared by value
var a:Boolean = Boolean("a"); // a is true
var b:Boolean = Boolean(1); // b is true
trace(a==b); // true

// Variables representing Boolean objects are compared by reference
var a:Boolean = new Boolean("a"); // a is true
var b:Boolean = new Boolean(1); // b is true
trace(a == b); // false
```

See also

`Boolean`

call function

```
call(frame)
```

Deprecated since Flash Player 5. This action was deprecated in favor of the `function` statement.

Executes the script in the called frame without moving the playhead to that frame. Local variables do not exist after the script executes.

- If variables are not declared inside a block ({}) but the action list was executed with a call() action, the variables are local and expire at the end of the current list.
- If variables are not declared inside a block and the current action list was not executed with the call() action, the variables are interpreted as Timeline variables.

Availability

Flash Lite 1.0

Parameters

`frame:Object` - The label or number of a frame in the Timeline.

See also

`Array function`, `call (Function.call method)`

chr function

```
chr(number) : String
```

Deprecated since Flash Player 5. This function was deprecated in favor of `String.fromCharCode()`.

Converts ASCII code numbers to characters.

Availability
Flash Lite 1.0

Parameters
`number:Number` - An ASCII code number.

Returns
`String` - The character value of the specified ASCII code.

Example
The following example converts the number 65 to the letter A and assigns it to the variable `myVar`: `myVar = chr(65);`

See also
`fromCharCode (String.fromCharCode method)`

clearInterval function

```
clearInterval(intervalID:Number) : Void
```

Cancels an interval created by a call to setInterval().

Availability
Flash Lite 2.0

Parameters
`intervalID:Number` - A numeric (integer) identifier returned from a call to `setInterval()`.

Example
The following example first sets and then clears an interval call:

```
function callback() {
    trace("interval called: "+getTimer()+" ms.");
}

var intervalID:Number = setInterval(callback, 1000);
```

You need to clear the interval when you have finished using the function. Create a button called `clearInt_btn` and use the following ActionScript to clear `setInterval()`:

```
clearInt_btn.onRelease = function(){
    clearInterval( intervalID );
    trace("cleared interval");
};
```

See also
`setInterval function`

duplicateMovieClip function

```
duplicateMovieClip(target:String, newname:String, depth:Number) : Void
duplicateMovieClip(target:MovieClip, newname:String, depth:Number) : Void
```

Creates an instance of a movie clip while the SWF file is playing. The playhead in duplicate movie clips always starts at Frame 1, regardless of where the playhead is in the original movie clip. Variables in the original movie clip are not copied into the duplicate movie clip. Use the `removeMovieClip()` function or method to delete a movie clip instance created with `duplicateMovieClip()`.

Availability
Flash Lite 2.0

Parameters
`target:Object` - The target path of the movie clip to duplicate. This parameter can be either a string (e.g. "my_mc") or a direct reference to the movie clip instance (e.g. my_mc). Parameters that can accept more than one data type are listed as type `Object`.

`newname:String` - A unique identifier for the duplicated movie clip.

`depth:Number` - A unique depth level for the duplicated movie clip. The depth level is a stacking order for duplicated movie clips. This stacking order is similar to the stacking order of layers in the Timeline; movie clips with a lower depth level are hidden under clips with a higher stacking order. You must assign each duplicated movie clip a unique depth level to prevent it from replacing SWF files on occupied depths.

Example
In the following example, a new movie clip instance is created called `img_mc`. An image is loaded into the movie clip, and then the `img_mc` clip is duplicated. The duplicated clip is called `newImg_mc`, and this new clip is moved on the Stage so it does not overlap the original clip, and the same image is loaded into the second clip.

```
this.createEmptyMovieClip("img_mc", this.getNextHighestDepth());
img_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
duplicateMovieClip(img_mc, "newImg_mc", this.getNextHighestDepth());
newImg_mc._x = 200;
newImg_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
```

To remove the duplicate movie clip, you could add this code for a button called `myButton_btn`.

```
this.myButton_btn.onRelease = function(){
    removeMovieClip(newImg_mc);
};
```

See also
`removeMovieClip function`, `duplicateMovieClip (MovieClip.duplicateMovieClip method)`, `removeMovieClip (MovieClip.removeMovieClip method)`

escape function

```
escape(expression:String) : String
```

Converts the parameter to a string and encodes it in a URL-encoded format, where all nonalphanumeric characters are replaced with % hexadecimal sequences. When used in a URL-encoded string, the percentage symbol (%) is used to introduce escape characters, and is not equivalent to the modulo operator (%).

Availability
Flash Lite 2.0

Parameters

`expression:String` - The expression to convert into a string and encode in a URL-encoded format.

Returns

`String` - URL-encoded string.

Example

The following code produces the result `someuser%40somedomain%2Ecom`:

```
var email:String = "someuser@somedomain.com";
trace(escape(email));
```

In this example, the at symbol (`@`) was replaced with `%40` and the dot symbol (`.`) was replaced with `%2E`. This is useful if you're trying to pass information to a remote server and the data has special characters in it (for example, `&` or `?`), as shown in the following code:

```
var redirectUrl = "http://www.somedomain.com?loggedin=true&username=Gus";
getURL("http://www.myothersite.com?returnurl="+ escape(redirectUrl));
```

See also

`unescape function`

eval function

`eval(expression:Object) : Objecteval(expression:String) : Object`

Accesses variables, properties, objects, or movie clips by name. If expression is a variable or a property, the value of the variable or property is returned. If expression is an object or movie clip, a reference to the object or movie clip is returned. If the element named in expression cannot be found, undefined is returned.

In Flash 4, `eval()` was used to simulate arrays; in Flash 5 or later, you should use the Array class to simulate arrays.

In Flash 4, you can also use `eval()` to dynamically set and retrieve the value of a variable or instance name. However, you can also do this with the array access operator (`[]`).

In Flash 5 or later, you cannot use `eval()` to dynamically set and retrieve the value of a variable or instance name, because you cannot use`eval()` on the left side of an equation. For example, replace the code

```
eval ("var" + i) = "first";
```

with this:

```
this["var"+i] = "first"
```

or this:

```
set ("var" + i, "first");
```

Availability

Flash Lite 1.0

Parameters

`expression:Object` - The name of a variable, property, object, or movie clip to retrieve. This parameter can be either a string or a direct reference to the object instance (i.e., use of quotation marks (" ") is optional.)

Returns

`Object` - A value, reference to an object or movie clip, or `undefined`.

Example

The following example uses `eval()` to set properties for dynamically named movie clips. This ActionScript sets the `_rotation` property for three movie clips, called `square1_mc`, `square2_mc`, and `square3_mc`.

```
for (var i = 1; i <= 3; i++) {
    setProperty(eval("square"+i+"_mc"), _rotation, 5);
}
```

You can also use the following ActionScript:

```
for (var i = 1; i <= 3; i++) {
    this["square"+i+"_mc"]._rotation = -5;
}
```

See also

`Array`, `set variable statement`

fscommand function

```
fscommand(command:String, parameters:String) : Void
```

The `fscommand()` function lets a SWF file communicate with the Flash Lite player or the environment for a mobile device (such as an operating system). The parameters define the name of the application being started and the parameters to it, separated by commas.

Command	Parameters	Purpose
`launch`	`application-path, arg1, arg2,..., argn`	This command launches another application on a mobile device. The name of the application and its parameters are passed in as a single argument. **Note**: This feature is operating-system dependent. Please use this command carefully as it can call on the host device to perform an unsupported operation. Using it in this way could cause the host device to crash. This command is supported only when the Flash Lite player is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).
`activateTextField`	`"" (ignored)`	This command asynchronously activates the currently selected text field, making it active for user edits. Because it behaves asynchronously, this command is processed at the end of the frame. ActionScript that immediately follows the call to `fscommand()` executes first. If no text field is selected when the command is processed, nothing happens. This command gives focus to a text field previously passed to the `Selection.setFocus()` method and activates the text field for editing. This command has an effect only when the handset supports inline text editing. This command can be called as part of the `Selection.onSetFocus()` event listener callback. This causes text fields to become active for editing when they are selected. **Note**: Because the `fscommand()` function is executed asynchronously, the text field does not immediately become active; it becomes active at the end of the frame.

Availability

Flash Lite 1.1

Parameters

`command:String` - A string passed to the host application for any use, or a command passed to the Flash Lite player.

`parameters:String` - A string passed to the host application for any use, or a value passed to the Flash Lite player.

Example

In the following example, the `fscommand()` function opens wap.yahoo.com on the services/Web browser on Series 60 phones:

```
on(keyPress "9") {
    status = fscommand("launch", "z:\\system\apps\browser\browser.app,http://wap.yahoo.com");
}
```

fscommand2 function

```
fscommand2(command:String, parameter1:String,...parameterN:String) : Void
```

Lets the SWF file communicate with the Flash Lite player or a host application on a mobile device.

To use `fscommand2()` to send a message to the Flash Lite player, you must use predefined commands and parameters. See the "fscommand2 Commands" section under "ActionScript language elements" for the values you can specify for the `fscommand()` function's commands and parameters. These values control SWF files that are playing in the Flash Lite player.

The `fscommand2()` function is similar to the `fscommand()` function, with the following differences:

- The `fscommand2()` function can take any number of arguments. By contrast, `fscommand()` can take only one argument.
- Flash Lite executes `fscommand2()` immediately (in other words, within the frame), whereas `fscommand()` is executed at the end of the frame being processed.
- The `fscommand2()` function returns a value that can be used to report success, failure, or the result of the command.

Note: None of the `fscommand2()` commands are available in Web players.

Availability

Flash Lite 1.1

Deprecated fscommand2() commands

Some `fscommand2()` commands from Flash Lite 1.1 have been deprecated in Flash Lite 2.0. The following table shows the deprecated `fscommand2()` commands:

Command	Deprecated By
Escape	`escape` global function
GetDateDay	`getDate()` method of Date object
GetDateMonth	`getMonth()` method of Date object
GetDateWeekday	`getDay()` method of Date object
GetDateYear	`getYear()` method of Date object
GetLanguage	`System.capabilities.language` property
GetLocaleLongDate	`getLocaleLongDate()` method of Date object
GetLocaleShortDate	`getLocaleShortDate()` method of Date object
GetLocaleTime	`getLocaleTime()` method of Date object
GetTimeHours	`getHours()` method of Date object
GetTimeMinutes	`getMinutes()` method of Date object
GetTimeSeconds	`getSeconds()` method of Date object
GetTimeZoneOffset	`getTimeZoneOffset()` method of Date object
SetQuality	`MovieClip._quality`
Unescape	`unescape()` global function

Parameters

`command:String` - A string passed to the host application for any use, or a command passed to the Flash Lite player.

`parameters:String` - A string passed to the host application for any use, or a value passed to the Flash Lite player.

getProperty function

```
getProperty(my_mc:Object, property:Object) : Object
```

Deprecated since Flash Player 5. This function was deprecated in favor of the dot syntax, which was introduced in Flash Player 5.

Returns the value of the specified property for the movie clip *my_mc*.

Availability
Flash Lite 1.0

Parameters
`my_mc:Object` - The instance name of a movie clip for which the property is being retrieved.

`property:Object` - A property of a movie clip.

Returns
`Object` - The value of the specified property.

Example
The following example creates a new movie clip `someClip_mc` and shows the alpha value (`_alpha`) for the movie clip `someClip_mc` in the Output panel:

```
this.createEmptyMovieClip("someClip_mc", 999);
trace("The alpha of "+getProperty(someClip_mc, _name)+" is: "+getProperty(someClip_mc,
_alpha));
```

getTimer function

```
getTimer() : Number
```

Returns the number of milliseconds that have elapsed since the SWF file started playing.

Availability
Flash Lite 1.0

Returns
`Number` - The number of milliseconds that have elapsed since the SWF file started playing.

Example
In the following example, the `getTimer()` and `setInterval()` functions are used to create a simple timer:

```
this.createTextField("timer_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
function updateTimer():Void {
    timer_txt.text = getTimer();
}

var intervalID:Number = setInterval(updateTimer, 100);
```

getURL function

```
getURL(url:String [, window:String [, method:String] ]) : Void
```

Loads a document from a specific URL into a window or passes variables to another application at a defined URL. To test this function, make sure the file to be loaded is at the specified location. To use an absolute URL (for example, *http://www.myserver.com*), you need a network connection.

Note: This function is not supported for BREW devices.

Availability
Flash Lite 1.0

Parameters
`url:String` - The URL from which to obtain the document.

`window:String` [optional] - Specifies the window or HTML frame into which the document should load. You can enter the name of a specific window or select from the following reserved target names:

- `_self` specifies the current frame in the current window.
- `_blank` specifies a new window.
- `_parent` specifies the parent of the current frame.
- `_top` specifies the top-level frame in the current window.

`method:String` [optional] - A `GET` or `POST` method for sending variables. If there are no variables, omit this parameter. The `GET` method appends the variables to the end of the URL, and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for sending long strings of variables.

Example
This example loads an image into a movie clip. When the image is clicked, a new URL is loaded in a new browser window.

```
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
    target_mc.onRelease = function() {
    getURL("http://www.macromedia.com/software/flash/flashpro/", "_blank");
    };
};
var logo:MovieClipLoader = new MovieClipLoader();
logo.addListener(listenerObject);
logo.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    this.createEmptyMovieClip("macromedia_mc", this.getNextHighestDepth()));
```

In the following example, `getURL()` is used to send an e-mail message:

```
myBtn_btn.onRelease = function(){
    getURL("mailto:you@somedomain.com");
};
```

You can also use `GET` or `POST` for sending variables. The following example uses `GET` to append variables to a URL:

```
var firstName:String = "Gus";
var lastName:String = "Richardson";
var age:Number = 92;
myBtn_btn.onRelease = function() {
    getURL("http://www.macromedia.com", "_blank", "GET");
};
```

The following ActionScript uses `POST` to send variables in the HTTP header. Make sure you test your documents in a browser window, because otherwise your variables are sent using `GET`:

```
var firstName:String = "Gus";
var lastName:String = "Richardson";
var age:Number = 92;
getURL("http://www.macromedia.com", "_blank", "POST");
```

See also

loadVariables function, send (XML.send method), sendAndLoad (XML.sendAndLoad method)

getVersion function

```
getVersion() : String
```

Returns a string containing Flash Player version and platform information. The `getVersion` function returns information only for Flash Player 5 or later versions of Flash Player.

Availability

Flash Lite 2.0

Returns

`String` - A string containing Flash Player version and platform information.

Example

The following examples trace the version number of the Flash Player playing the SWF file:

```
var flashVersion:String = getVersion();
trace(flashVersion); // output: WIN 8,0,1,0
trace($version); // output: WIN 8,0,1,0
trace(System.capabilities.version); // output: WIN 8,0,1,0
```

The following string is returned by the `getVersion` function:

```
WIN 8,0,1,0
```

This returned string indicates that the platform is Microsoft Windows, and the version number of Flash Player is major version 8, minor version 1 (8.1).

See also

os (capabilities.os property), version (capabilities.version property)

gotoAndPlay function

```
gotoAndPlay( [scene:String,] frame:Object) : Void
```

Sends the playhead to the specified frame in a scene and plays from that frame. If no scene is specified, the playhead goes to the specified frame in the current scene. You can use the *scene* parameter only on the root Timeline, not within Timelines for movie clips or other objects in the document.

Availability

Flash Lite 1.0

Parameters

`scene:String` [optional] - A string specifying the name of the scene to which the playhead is sent.

`frame:Object` - A number representing the frame number, or a string representing the label of the frame, to which the playhead is sent.

Example

In the following example, a document has two scenes: `sceneOne` and `sceneTwo`. Scene one contains a frame label on Frame 10 called `newFrame` and two buttons, `myBtn_btn` and `myOtherBtn_btn`. This ActionScript is placed on Frame 1, Scene 1 of the main Timeline.

```
stop();
myBtn_btn.onRelease = function(){
    gotoAndPlay("newFrame");
};

myOtherBtn_btn.onRelease = function(){
    gotoAndPlay("sceneTwo", 1);
};
```

When the user clicks the buttons, the playhead moves to the specified location and continues playing.

See also

`gotoAndPlay (MovieClip.gotoAndPlay method)`, `nextFrame function`, `play function`, `prevFrame function`

gotoAndStop function

```
gotoAndStop( [scene:String,] frame:Object) : Void
```

Sends the playhead to the specified frame in a scene and stops it. If no scene is specified, the playhead is sent to the frame in the current scene.You can use the *scene* parameter only on the root Timeline, not within Timelines for movie clips or other objects in the document.

Availability

Flash Lite 1.0

Parameters

`scene:String` [optional] - A string specifying the name of the scene to which the playhead is sent.

`frame:Object` - A number representing the frame number, or a string representing the label of the frame, to which the playhead is sent.

Example

In the following example, a document has two scenes: `sceneOne` and `sceneTwo`. Scene one contains a frame label on Frame 10 called `newFrame`, and two buttons, `myBtn_btn` and `myOtherBtn_btn`. This ActionScript is placed on Frame 1, Scene 1 of the main Timeline:

```
stop();

myBtn_btn.onRelease = function(){
    gotoAndStop("newFrame");
};

myOtherBtn_btn.onRelease = function(){
    gotoAndStop("sceneTwo", 1);
};
```

When the user clicks the buttons, the playhead moves to the specified location and stops.

See also

gotoAndStop (MovieClip.gotoAndStop method), stop function, play function, gotoAndPlay function

ifFrameLoaded function

```
ifFrameLoaded( [scene,] frame) { statement(s); }
```

Deprecated since Flash Player 5. This function has been deprecated. Adobe recommends that you use the `MovieClip._framesloaded` property.

Checks whether the contents of a specific frame are available locally. Use `ifFrameLoaded` to start playing a simple animation while the rest of the SWF file downloads to the local computer. The difference between using `_framesloaded` and `ifFrameLoaded` is that `_framesloaded` lets you add custom `if` or `else` statements.

Availability

Flash Lite 1.0

Parameters

`scene:String` [optional] - A string that specifies the name of the scene that must be loaded.

`frame:Object` - The frame number or frame label that must be loaded before the next statement is executed.

`statement(s):Object` - The instructions to execute if the specified scene, or scene and frame, are loaded.

See also

addListener (MovieClipLoader.addListener method)

int function

```
int(value) : Number
```

Deprecated since Flash Player 5. This function was deprecated in favor of `Math.round()`.

Converts a decimal number to an integer value by truncating the decimal value. This function is equivalent to `Math.floor()` if the *value* parameter is positive and `Math.ceil()` if the *value* parameter is negative.

Availability

Flash Lite 1.0

Parameters

`value:Number` - A number to be rounded to an integer.

Returns

`Number` - The truncated integer value.

See also

round (Math.round method), floor (Math.floor method), ceil (Math.ceil method)

isFinite function

```
isFinite(expression:Object) : Boolean
```

Evaluates *expression* and returns `true` if it is a finite number or `false` if it is infinity or negative infinity. The presence of infinity or negative infinity indicates a mathematical error condition such as division by 0.

Availability

Flash Lite 2.0

Parameters

`expression:Object` - A Boolean value, variable, or other expression to be evaluated.

Returns

`Boolean` - A Boolean value.

Example

The following example shows return values for `isFinite`:

```
isFinite(56)
// returns true

isFinite(Number.POSITIVE_INFINITY)
//returns false
```

isNaN function

```
isNaN(expression:Object) : Boolean
```

Evaluates the parameter and returns `true` if the value is `NaN`(not a number). This function is useful for checking whether a mathematical expression evaluates successfully to a number.

Availability

Flash Lite 2.0

Parameters

`expression:Object` - A Boolean, variable, or other expression to be evaluated.

Returns

`Boolean` - A Boolean value.

Example

The following code illustrates return values for the `isNaN()` function:

```
trace( isNaN("Tree") );
// returns true

trace( isNaN(56) );
// returns false

trace( isNaN(Number.POSITIVE_INFINITY) )
// returns false
```

The following example shows how you can use `isNAN()` to check whether a mathematical expression contains an error:

```
var dividend:Number;
var divisor:Number;
divisor = 1;
trace( isNaN(dividend/divisor) );
// output: true
// The output is true because the variable dividend is undefined.
// Do not use isNAN() to check for division by 0 because it will return false.
// A positive number divided by 0 equals Infinity (Number.POSITIVE_INFINITY).
// A negative number divided by 0 equals -Infinity (Number.NEGATIVE_INFINITY).
```

See also

`NaN constant`, `NaN (Number.NaN property)`

length function

```
length(expression)length(variable)
```

Deprecated since Flash Player 5. This function, along with all the string functions, has been deprecated. Adobe recommends that you use the methods of the String class and the `String.length` property to perform the same operations.

Returns the length of the specified string or variable.

Availability

Flash Lite 1.0

Parameters

`expression:String` - A string.

`variable:Object` - The name of a variable.

Returns

`Number` - The length of the specified string or variable.

Example

The following example returns the length of the string "Hello": `length("Hello");` The result is 5.

See also

`" string delimiter operator`, `String`, `length (String.length property)`

loadMovie function

```
loadMovie(url:String, target:Object [, method:String]) : Void
loadMovie(url:String, target:String [, method:String]) : Void
```

Loads a SWF or JPEG file into Flash Player while the original SWF file plays. JPEG files saved in progressive format are not supported.

If you want to monitor the progress of the download, use `MovieClipLoader.loadClip()` instead of this function.

The `loadMovie()` function lets you display several SWF files at once and switch among SWF files without loading another HTML document. Without the `loadMovie()` function, Flash Player displays a single SWF file.

If you want to load a SWF or JPEG file into a specific level, use `loadMovieNum()` instead of `loadMovie()`.

When a SWF file is loaded into a target movie clip, you can use the target path of that movie clip to target the loaded SWF file. A SWF file or image loaded into a target inherits the position, rotation, and scale properties of the targeted movie clip. The upper left corner of the loaded image or SWF file aligns with the registration point of the targeted movie clip. Alternatively, if the target is the root Timeline, the upper left corner of the image or SWF file aligns with the upper left corner of the Stage.

Use `unloadMovie()` to remove SWF files that were loaded with `loadMovie()`.

Availability
Flash Lite 1.1

Parameters
`url:String` - The absolute or relative URL of the SWF or JPEG file to be loaded. A relative path must be relative to the SWF file at level 0. Absolute URLs must include the protocol reference, such as http:// or file:///.

`target:Object` - A reference to a movie clip object or a string representing the path to a target movie clip. The target movie clip is replaced by the loaded SWF file or image.

`method:String` [optional] - Specifies an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Example
Usage 1: The following example loads the SWF file circle.swf from the same directory and replaces a movie clip called `mySquare` that already exists on the Stage:

```
loadMovie("circle.swf", mySquare);
// equivalent statement (Usage 1): loadMovie("circle.swf", _level0.mySquare);
// equivalent statement (Usage 2): loadMovie("circle.swf", "mySquare");
```

The following example loads the SWF file circle.swf from the same directory, but replaces the main movie clip instead of the `mySquare` movie clip:

```
loadMovie("circle.swf", this);
// Note that using "this" as a string for the target parameter will not work
// equivalent statement (Usage 2): loadMovie("circle.swf", "_level0");
```

The following `loadMovie()` statement loads the SWF file sub.swf from the same directory into a new movie clip called `logo_mc` that's created using `createEmptyMovieClip()`:

```
this.createEmptyMovieClip("logo_mc", 999);
loadMovie("sub.swf", logo_mc);
```

You could add the following code to load a JPEG image called image1.jpg from the same directory as the SWF file loading sub.swf. The JPEG is loaded when you click a button called `myBtn_btn`. This code loads the JPEG into `logo_mc`. Therefore, it will replace sub.swf with the JPEG image.

```
myBtn_btn.onRelease = function(){
    loadMovie("image1.jpg", logo_mc);
};
```

Usage 2: The following example loads the SWF file circle.swf from the same directory and replaces a movie clip called `mySquare` that already exists on the Stage:

```
loadMovie("circle.swf", "mySquare");
```

See also

`_level property`, `loadMovieNum function`, `loadMovie (MovieClip.loadMovie method)`, `loadClip (MovieClipLoader.loadClip method)unloadMovie function`

loadMovieNum function

```
loadMovieNum(url:String, level:Number [, method:String]) : Void
```

Loads a SWF or JPEG file into a level in Flash Player while the originally loaded SWF file plays.

If you want to monitor the progress of the download, use `MovieClipLoader.loadClip()` instead of this function.

Normally, Flash Player displays a single SWF file and then closes. The `loadMovieNum()` action lets you display several SWF files at once and switch among SWF files without loading another HTML document.

If you want to specify a target instead of a level, use `loadMovie()` instead of `loadMovieNum()`.

Flash Player has a stacking order of levels starting with level 0. These levels are like layers of acetate; they are transparent except for the objects on each level. When you use `loadMovieNum()`, you must specify a level in Flash Player into which the SWF file will load. When a SWF file is loaded into a level, you can use the syntax, `_level`*N*, where *N* is the level number, to target the SWF file.

When you load a SWF file, you can specify any level number and you can load SWF files into a level that already has a SWF file loaded into it. If you do, the new SWF file will replace the existing SWF file. If you load a SWF file into level 0, every level in Flash Player is unloaded, and level 0 is replaced with the new file. The SWF file in level 0 sets the frame rate, background color, and frame size for all other loaded SWF files.

The `loadMovieNum()` action also lets you load JPEG files into a SWF file while it plays. For images and SWF files, the upper left corner of the image aligns with the upper left corner of the Stage when the file loads. Also in both cases, the loaded file inherits rotation and scaling, and the original content is overwritten in the specified level.

Note: *JPEG files saved in progressive format are not supported.*

Use `unloadMovieNum()` to remove SWF files or images that were loaded with `loadMovieNum()`.

Availability

Flash Lite 1.1

Parameters

`url:String` - The absolute or relative URL of the SWF or JPEG file to be loaded. A relative path must be relative to the SWF file at level 0. For use in the stand-alone Flash Player or for testing in test mode in the Flash authoring application, all SWF files must be stored in the same folder and the filenames cannot include folder or disk drive specifications.

`level:Number` - An integer specifying the level in Flash Player into which the SWF file will load.

`method:String` [optional] - Specifies an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Example

The following example loads the JPEG image tim.jpg into level 2 of Flash Player:

```
loadMovieNum("http://www.helpexamples.com/flash/images/image1.jpg", 2);
```

See also

`unloadMovieNum function`, `loadMovie function`, `loadClip (MovieClipLoader.loadClip method)`, `_level property`

loadVariables function

```
loadVariables(url:String, target:Object [, method:String]) : Void
```

Reads data from an external file, such as a text file or text generated by ColdFusion, a CGI script, Active Server Pages (ASP), PHP, or Perl script, and sets the values for variables in a target movie clip. This action can also be used to update variables in the active SWF file with new values.

The text at the specified URL must be in the standard MIME format *application/x-www-form-urlencoded* (a standard format used by CGI scripts). Any number of variables can be specified. For example, the following phrase defines several variables:

```
company=Macromedia&address=600+Townsend&city=San+Francisco&zip=94103
```

In SWF files running in a version earlier than Flash Player 7, *url* must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the leftmost component of a file's URL. For example, a SWF file at www.someDomain.com can load data from a source at store.someDomain.com because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, *url* must be in exactly the same domain as the SWF file that is issuing this call (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file that is being accessed. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

If you want to load variables into a specific level, use `loadVariablesNum()` instead of `loadVariables()`.

Availability

Flash Lite 1.1

Parameters

`url:String` - An absolute or relative URL where the variables are located. If the SWF file issuing this call is running in a web browser, *url* must be in the same domain as the SWF file; for details, see the Description section.

`target:Object` - The target path to a movie clip that receives the loaded variables.

`method:String` [optional] - Specifies an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Example

The following example loads information from a text file called params.txt into the `target_mc` movie clip that is created using `createEmptyMovieClip()`. The `setInterval()` function is used to check the loading progress. The script checks for a variable in the params.txt file named `done`.

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
loadVariables("params.txt", target_mc);
function checkParamsLoaded() {
    if (target_mc.done == undefined) {
    trace("not yet.");
    } else {
    trace("finished loading. killing interval.");
    trace("-------------");
    for (i in target_mc) {
    trace(i+": "+target_mc[i]);
    }
    trace("-------------");
    clearInterval(param_interval);
    }
}
var param_interval = setInterval(checkParamsLoaded, 100);
```

The external file, `params.txt`, includes the following text:

```
var1="hello"&var2="goodbye"&done="done"
```

See also

loadVariablesNum function, loadMovie function, loadMovieNum function, getURL function, loadMovie (MovieClip.loadMovie method) loadVariables (MovieClip.loadVariables method), load (LoadVars.load method)

loadVariablesNum function

```
loadVariablesNum(url:String, level:Number [, method:String]) : Void
```

Reads data from an external file, such as a text file or text generated by ColdFusion, a CGI script, ASP, PHP, or a Perl script, and sets the values for variables in a Flash Player level. You can also use this function to update variables in the active SWF file with new values.

The text at the specified URL must be in the standard MIME format *application/x-www-form-urlencoded* (a standard format used by CGI scripts). Any number of variables can be specified. For example, the following phrase defines several variables:

```
company=Macromedia&address=601+Townsend&city=San+Francisco&zip=94103
```

In SWF files running in a version of the player earlier than Flash Player 7, *url* must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the leftmost component of a file's URL. For example, a SWF file at www.someDomain.com can load data from a source at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, *url* must be in exactly the same domain as the SWF file that is issuing this call (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

If you want to load variables into a target MovieClip, use `loadVariables()` instead of `loadVariablesNum()`.

Availability

Flash Lite 1.1

Parameters

`url:String` - An absolute or relative URL where the variables are located. If the SWF file issuing this call is running in a web browser, *url* must be in the same domain as the SWF file; for details, see the Description section.

`level:Number` - An integer specifying the level in Flash Player to receive the variables.

`method:String` [optional] - Specifies an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Example

The following example loads information from a text file called params.txt into the main Timeline of the SWF at level 2 in Flash Player. The variable names of the text fields must match the variable names in the params.txt file. The `setInterval()` function is used to check the progress of the data being loaded into the SWF. The script checks for a variable in the params.txt file named `done`.

```
loadVariablesNum("params.txt", 2);
function checkParamsLoaded() {
    if (_level2.done == undefined) {
    trace("not yet.");
    } else {
    trace("finished loading. killing interval.");
    trace("-------------");
    for (i in _level2) {
    trace(i+": "+_level2[i]);
    }
    trace("-------------");
    clearInterval(param_interval);
    }
}
var param_interval = setInterval(checkParamsLoaded, 100);

// Params.txt includes the following text
var1="hello"&var2="goodbye"&done="done"
```

See also

`getURL function`, `loadMovie function`, `loadMovieNum function`, `loadVariables function`, `loadMovie (MovieClip.loadMovie method)` `loadVariables (MovieClip.loadVariables method)`, `load (LoadVars.load method)`

mbchr function

`mbchr(number)`

Deprecated since Flash Player 5. This function was deprecated in favor of the `String.fromCharCode()` method.

Converts an ASCII code number to a multibyte character.

Availability

Flash Lite 1.0

Parameters

`number:Number` - The number to convert to a multibyte character.

See also

`fromCharCode (String.fromCharCode method)`

mblength function

`mblength(string) : Number`

Deprecated since Flash Player 5. This function was deprecated in favor of the `String.length` property.

Returns the length of the multibyte character string.

Availability

Flash Lite 1.0

Parameters

`string:String` - The string to measure.

Returns

`Number` - The length of the multibyte character string.

See also

`String`, `length (String.length property)`

mbord function

`mbord(character) : Number`

Deprecated since Flash Player 5. This function was deprecated in favor of `String.charCodeAt()` method.

Converts the specified character to a multibyte number.

Availability

Flash Lite 1.0

Parameters

`character:String` - The character to convert to a multibyte number.

Returns

`Number` - The converted character.

See also

`charCodeAt (String.charCodeAt method)`

mbsubstring function

`mbsubstring(value, index, count) : String`

Deprecated since Flash Player 5. This function was deprecated in favor of `String.substr()` method.

Extracts a new multibyte character string from a multibyte character string.

Availability
Flash Lite 1.0

Parameters
`value:String` - The multibyte string from which to extract a new multibyte string.

`index:Number` - The number of the first character to extract.

`count:Number` - The number of characters to include in the extracted string, not including the index character.

Returns
`String` - The string extracted from the multibyte character string.

See also
`substr (String.substr method)`

nextFrame function

```
nextFrame() : Void
```

Sends the playhead to the next frame.

Availability
Flash Lite 1.0

Example
In the following example, when the user presses the Right or Down Arrow key, the playhead goes to the next frame and stops. If the user presses the Left or Up Arrow key, the playhead goes to the previous frame and stops. The listener is initialized to wait for the arrow key to be pressed, and the `init` variable is used to prevent the listener from being redefined if the playhead returns to Frame 1.

```
stop();

if (init == undefined) {
    someListener = new Object();
    someListener.onKeyDown = function() {
    if (Key.isDown(Key.LEFT) || Key.isDown(Key.UP)) {
    _level0.prevFrame();
    } else if (Key.isDown(Key.RIGHT) || Key.isDown(Key.DOWN)) {
    _level0.nextFrame();
    }
    };
    Key.addListener(someListener);
    init = 1;
}
```

See also
`prevFrame function`

nextScene function

```
nextScene() : Void
```

Sends the playhead to Frame 1 of the next scene.

Availability
Flash Lite 1.0

Example
In the following example, when a user clicks the button that is created at runtime, the playhead is sent to Frame 1 of the next scene. Create two scenes, and enter the following ActionScript on Frame 1 of Scene 1.

```
stop();

if (init == undefined) {
    this.createEmptyMovieClip("nextscene_mc", this.getNextHighestDepth());
    nextscene_mc.createTextField("nextscene_txt", this.getNextHighestDepth(), 200, 0, 100,
22);
    nextscene_mc.nextscene_txt.autoSize = true;
    nextscene_mc.nextscene_txt.border = true;
    nextscene_mc.nextscene_txt.text = "Next Scene";
    this.createEmptyMovieClip("prevscene_mc", this.getNextHighestDepth());
    prevscene_mc.createTextField("prevscene_txt", this.getNextHighestDepth(), 00, 0, 100, 22);
    prevscene_mc.prevscene_txt.autoSize = true;
    prevscene_mc.prevscene_txt.border = true;
    prevscene_mc.prevscene_txt.text = "Prev Scene";
    nextscene_mc.onRelease = function() {
    nextScene();
    };

    prevscene_mc.onRelease = function() {
    prevScene();
    };

    init = true;
}
```

Make sure you place a `stop()` action on Frame 1 of Scene 2.

See also
`prevScene function`

Number function

```
Number(expression) : Number
```

Converts the parameter *expression* to a number and returns a value as described in the following list:

- If *expression* is a number, the return value is *expression*.
- If *expression* is a Boolean value, the return value is 1 if *expression* is `true`, 0 if *expression* is `false`.
- If *expression* is a string, the function attempts to parse *expression* as a decimal number with an optional trailing exponent (that is, 1.57505e-3).
- If *expression* is `NaN`, the return value is `NaN`.
- If *expression* is `undefined`, the return value is as follows:
- - In files published for Flash Player 6 or earlier, the result is 0.

- - In files published for Flash Player 7 or later, the result is `NaN`.

Availability

Flash Lite 2.0

Parameters

`expression:Object` - An expression to convert to a number. Numbers or strings that begin with 0x are interpreted as hexadecimal values. Numbers or strings that begin with 0 are interpreted as octal values.

Returns

`Number` - A number or `NaN` (not a number).

Example

In the following example, a text field is created on the Stage at runtime:

```
this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
counter_txt.autoSize = true;
counter_txt.text = 0;
function incrementInterval():Void {
    var counter:Number = counter_txt.text;
    // Without the Number() function, Flash would concatenate the value instead
    // of adding values. You could also use "counter_txt.text++;"
    counter_txt.text = Number(counter) + 1;
}
var intervalID:Number = setInterval(incrementInterval, 1000);
```

See also

`NaN constant`, `Number`, `parseInt function`, `parseFloat function`

Object function

```
Object( [value] ) : Object
```

Creates a new empty object or converts the specified number, string, or Boolean value to an object. This command is equivalent to creating an object using the Object constructor (see "Constructor for the Object class").

Availability

Flash Lite 2.0

Parameters

`value:Object` [optional] - A number, string, or Boolean value.

Returns

`Object` - An object.

Example

In the following example, a new empty object is created, and then the object is populated with values:

```
var company:Object = new Object();
company.name = "Macromedia, Inc.";
company.address = "600 Townsend Street";
company.city = "San Francisco";
company.state = "CA";
company.postal = "94103";
for (var i in company) {
    trace("company."+i+" = "+company[i]);
}
```

See also

`Object`

on handler

```
on(mouseEvent:Object) { // your statements here }
```

Specifies the mouse event or keypress that triggers an action.

Availability

Flash Lite 2.0

Parameters

`mouseEvent:Object` - A *mouseEvent* is a trigger called an *event*. When the event occurs, the statements following it within curly braces ({ }) execute. Any of the following values can be specified for the *mouseEvent* parameter:

- `press` The mouse button is pressed while the pointer is over the button.
- `release` The mouse button is released while the pointer is over the button.
- `releaseOutside` While the pointer is over the button, the mouse button is pressed, rolled outside the button area, and released. Both the `press` and the `dragOut` events always precede a `releaseOutside` event. (This event is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.)
- `rollOut` The pointer rolls outside the button area. (This event is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.)
- `rollOver` The mouse pointer rolls over the button.
- `dragOut` While the pointer is over the button, the mouse button is pressed and then rolls outside the button area.
- `dragOver` While the pointer is over the button, the mouse button has been pressed, then rolled outside the button, and then rolled back over the button.
- `keyPress` "<*key*> " The specified keyboard key is pressed. For the *key* portion of the parameter, specify a key constant, as shown in the code hinting in the Actions panel. You can use this parameter to intercept a key press, that is, to override any built-in behavior for the specified key. The button can be anywhere in your application, on or off the Stage. One limitation of this technique is that you can't apply the `on()` handler at runtime; you must do it at authoring time. Make sure that you select Control > Disable Keyboard Shortcuts, or certain keys with built-in behavior won't be overridden when you test the application using Control > Test Movie.

For a list of key constants, see the Key class.

Example

In the following script, the `startDrag()` function executes when the mouse is pressed, and the conditional script is executed when the mouse is released and the object is dropped:

```
on (press) {
    startDrag(this);
}
on (release) {
    trace("X:"+this._x);
    trace("Y:"+this._y);
    stopDrag();
}
```

See also

`onClipEvent handler`, `Key`

onClipEvent handler

```
onClipEvent(movieEvent:Object) { // your statements here }
```

Triggers actions defined for a specific instance of a movie clip.

Availability

Flash Lite 2.0

Parameters

`movieEvent:Object` - The *movieEvent* is a trigger called an *event*. When the event occurs, the statements following it within curly braces ({}) are executed. Any of the following values can be specified for the *movieEvent* parameter:

- `load` The action is initiated as soon as the movie clip is instantiated and appears in the Timeline.
- `unload` The action is initiated in the first frame after the movie clip is removed from the Timeline. The actions associated with the `Unload` movie clip event are processed before any actions are attached to the affected frame.
- `enterFrame` The action is triggered continually at the frame rate of the movie clip. The actions associated with the `enterFrame` clip event are processed before any frame actions that are attached to the affected frames.
- `mouseMove` The action is initiated every time the mouse is moved. Use the `_xmouse` and `_ymouse` properties to determine the current mouse position. (This event is supported in Flash Lite only if `System.capabilities.hasMouse` is `true`.)
- `mouseDown` The action is initiated when the left mouse button is pressed. (This event is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.)
- `mouseUp` The action is initiated when the left mouse button is released. (This event is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.)
- `keyDown` The action is initiated when a key is pressed. Use `Key.getCode()` to retrieve information about the last key pressed.
- `keyUp` The action is initiated when a key is released. Use the `Key.getCode()` method to retrieve information about the last key pressed.
- `data` The action is initiated when data is received in a `loadVariables()` or `loadMovie()` action. When specified with a `loadVariables()` action, the `data` event occurs only once, when the last variable is loaded. When specified with a `loadMovie()` action, the `data` event occurs repeatedly, as each section of data is retrieved.

Example

The following example uses `onClipEvent()` with the `keyDown` movie event and is designed to be attached to a movie clip or button. The `keyDown` movie event is usually used with one or more methods and properties of the Key object. The following script uses `Key.getCode()` to find out which key the user has pressed; if the pressed key matches the `Key.RIGHT` property, the playhead is sent to the next frame; if the pressed key matches the `Key.LEFT` property, the playhead is sent to the previous frame.

```
onClipEvent (keyDown) {
    if (Key.getCode() == Key.RIGHT) {
    this._parent.nextFrame();
    } else if (Key.getCode() == Key.LEFT) {
    this._parent.prevFrame();
    }
}
```

The following example uses `onClipEvent()` with the `load` and `mouseMove` movie events. The `_xmouse` and `_ymouse` properties track the position of the mouse each time the mouse moves, which appears in the text field that's created at runtime.

```
onClipEvent (load) {
    this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
    coords_txt.autoSize = true;
    coords_txt.selectable = false;
}
onClipEvent (mouseMove) {
    coords_txt.text = "X:"+_root._xmouse+",Y:"+_root._ymouse;
}
```

See also

Key, _xmouse (MovieClip._xmouse property), _ymouse (MovieClip._ymouse property), Constants

ord function

```
ord(character) : Number
```

Deprecated since Flash Player 5. This function was deprecated in favor of the methods and properties of the String class.

Converts characters to ASCII code numbers.

Availability

Flash Lite 1.0

Parameters

`character:String` - The character to convert to an ASCII code number.

Returns

`Number` - The ASCII code number of the specified character.

See also

String, charCodeAt (String.charCodeAt method)

parseFloat function

`parseFloat(string:String) : Number`

Converts a string to a floating-point number. The function reads, or *parses*, and returns the numbers in a string until it reaches a character that is not a part of the initial number. If the string does not begin with a number that can be parsed, `parseFloat()` returns `NaN`. White space preceding valid integers is ignored, as are trailing nonnumeric characters.

Availability

Flash Lite 2.0

Parameters

`string:String` - The string to read and convert to a floating-point number.

Returns

`Number` - A number or `NaN` (not a number).

Example

The following examples use the `parseFloat()` function to evaluate various types of numbers:

```
trace(parseFloat("-2")); // output: -2
trace(parseFloat("2.5")); // output: 2.5
trace(parseFloat(" 2.5")); // output: 2.5
trace(parseFloat("3.5e6")); // output: 3500000
trace(parseFloat("foobar")); // output: NaN
trace(parseFloat("3.75math")); // output: 3.75
trace(parseFloat("0garbage")); // output: 0
```

See also

`NaN constant`, `parseInt function`

parseInt function

`parseInt(expression:String [, radix:Number]) : Number`

Converts a string to an integer. If the specified string in the parameters cannot be converted to a number, the function returns `NaN`. Strings beginning with 0x are interpreted as hexadecimal numbers. Integers beginning with 0 or specifying a radix of 8 are interpreted as octal numbers. White space preceding valid integers is ignored, as are trailing nonnumeric characters.

Availability

Flash Lite 2.0

Parameters

`expression:String` - A string to convert to an integer.

`radix:Number` [optional] - An integer representing the radix (base) of the number to parse. Legal values are from 2 to 36.

Returns

`Number` - A number or `NaN` (not a number).

Example

The examples in this section use the `parseInt()` function to evaluate various types of numbers.

The following example returns 3:

```
parseInt("3.5")
```

The following example returns `NaN`:

```
parseInt("bar")
```

The following example returns 4:

```
parseInt("4foo")
```

The following example shows a hexadecimal conversion that returns 1016:

```
parseInt("0x3F8")
```

The following example shows a hexadecimal conversion using the optional *radix* parameter that returns 1000:

```
parseInt("3E8", 16)
```

The following example shows a binary conversion and returns 10, which is the decimal representation of the binary 1010:

```
parseInt("1010", 2)
```

The following examples show octal number parsing and return 511, which is the decimal representation of the octal 777:

```
parseInt("0777")
parseInt("777", 8)
```

See also

NaN constant, parseFloat function

play function

```
play() : Void
```

Moves the playhead forward in the Timeline.

Availability

Flash Lite 1.0

Example

In the following example, there are two movie clip instances on the Stage with the instance names `stop_mc` and `play_mc`. The ActionScript stops the SWF file's playback when the `stop_mc` movie clip instance is clicked. Playback resumes when the `play_mc` instance is clicked.

```
this.stop_mc.onRelease = function() {
    stop();
};
this.play_mc.onRelease = function() {
    play();
};
trace("frame 1");
```

See also

gotoAndPlay function, gotoAndPlay (MovieClip.gotoAndPlay method)

prevFrame function

```
prevFrame() : Void
```

Sends the playhead to the previous frame. If the current frame is Frame 1, the playhead does not move.

Availability
Flash Lite 1.0

Example
When the user clicks a button called `myBtn_btn` and the following ActionScript is placed on a frame in the Timeline for that button, the playhead is sent to the previous frame:

```
stop();
this.myBtn_btn.onRelease = function(){
    prevFrame();
};
```

See also
`nextFrame function`, `prevFrame (MovieClip.prevFrame method)`

prevScene function

```
prevScene() : Void
```

Sends the playhead to Frame 1 of the previous scene.

Availability
Flash Lite 1.0

See also
`nextScene function`

random function

```
random(value) : Number
```

Deprecated since Flash Player 5. This function was deprecated in favor of `Math.random()`.

Returns a random integer between 0 and one less than the integer specified in the *value* parameter.

Availability
Flash Lite 1.1

Parameters
`value:Number` - An integer.

Returns
`Number` - A random integer.

Example
The following use of `random()` returns a value of 0, 1, 2, 3, or 4: `random(5);`

See also

random (Math.random method)

removeMovieClip function

```
removeMovieClip(target:Object)
```

Deletes the specified movie clip.

Availability

Flash Lite 1.0

Parameters

`target:Object` - The target path of a movie clip instance created with `duplicateMovieClip()` or the instance name of a movie clip created with `MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()`, or `MovieClip.createEmptyMovieClip()`.

Example

The following example creates a new movie clip called `myClip_mc` and duplicates the movie clip. The second movie clip is called `newClip_mc`. Images are loaded into both movie clips. When a button, `button_mc`, is clicked, the duplicated movie clip is removed from the Stage.

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());
myClip_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
duplicateMovieClip(this.myClip_mc, "newClip_mc", this.getNextHighestDepth());
newClip_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
newClip_mc._x = 200;
this.button_mc.onRelease = function() {
    removeMovieClip(this._parent.newClip_mc);
};
```

See also

duplicateMovieClip function, duplicateMovieClip (MovieClip.duplicateMovieClip method), attachMovie (MovieClip.attachMovie method), removeMovieClip (MovieClip.removeMovieClip method)createEmptyMovieClip (MovieClip.createEmptyMovieClip method)

setInterval function

```
setInterval(functionName:Object, interval:Number [, param1:Object, param2, ..., paramN]) :
Number
setInterval(objectName:Object, methodName:String, interval:Number [, param1:Object, param2,
..., paramN]) : Number
```

Calls a function or a method or an object at periodic intervals while a SWF file plays. You can use an interval function to update variables from a database or to update a time display.

If *interval* is greater than the SWF file's frame rate, the interval function is only called each time the playhead enters a frame; this minimizes the impact each time the screen is refreshed.

Note: In Flash Lite 2.0, the interval passed into this method is ignored if it is less than the SWF file's frame rate and the interval function is called on the SWF file's frame rate interval only. If the interval is greater than the SWF file's frame rate, the event is called on the next frame after the interval has elapsed.

Availability
Flash Lite 2.0

Parameters
`functionName:Object` - A function name or a reference to an anonymous function.

`interval:Number` - The time in milliseconds between calls to the *functionName* or *methodName* parameter.

`param:Object` [optional] - Parameters passed to the *functionName* or *methodName* parameter. Multiple parameters should be separated by commas: *param1*, *param2*, . . . , *paramN*.

`objectName:Object` - An object containing the method *methodName*.

`methodName:String` - A method of *objectName*.

Returns
`Number` - An identifying integer that you can pass to `clearInterval()` to cancel the interval.

Example
Usage 1: The following example calls an anonymous function every 1000 milliseconds (1 second).

```
setInterval( function(){ trace("interval called"); }, 1000 );
```

Usage 2: The following example defines two event handlers and calls each of them. The first call to `setInterval()` calls the `callback1()` function, which contains a `trace()` statement. The second call to `setInterval()` passes the `"interval called"` string to the function `callback2()` as a parameter.

```
function callback1() {
    trace("interval called");
}

function callback2(arg) {
    trace(arg);
}

setInterval( callback1, 1000 );
setInterval( callback2, 1000, "interval called" );
```

Usage 3: This example uses a method of an object. You must use this syntax when you want to call a method that is defined for an object.

```
obj = new Object();
obj.interval = function() {
    trace("interval function called");
}

setInterval( obj, "interval", 1000 );

obj2 = new Object();
obj2.interval = function(s) {
    trace(s);
}
setInterval( obj2, "interval", 1000, "interval function called" );
```

You must use the second form of the `setInterval()` syntax to call a method of an object, as shown in the following example:

```
setInterval( obj2, "interval", 1000, "interval function called" );
```

When working with this function, you need to be careful about the memory you use in a SWF file. For example, when you remove a movie clip from the SWF file, it will not remove any `setInterval()` function running within it. Always remove the `setInterval()` function by using `clearInterval()` when you have finished using it, as shown in the following example:

```
// create an event listener object for our MovieClipLoader instance
var listenerObjectbject = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
    trace("start interval");
    /* after the target movie clip loaded, create a callback which executes
    about every 1000 ms (1 second) and calls the intervalFunc function. */
    target_mc.myInterval = setInterval(intervalFunc, 1000, target_mc);
};
function intervalFunc(target_mc) {
    // display a trivial message which displays the instance name and arbitrary text.
    trace(target_mc+" has been loaded for "+getTimer()/1000+" seconds.");
    /* when the target movie clip is clicked (and released) you clear the interval
    and remove the movie clip. If you don't clear the interval before deleting
    the movie clip, the function still calls itself every second even though the
    movie clip instance is no longer present. */
    target_mc.onRelease = function() {
    trace("clear interval");
    clearInterval(this.myInterval);
    // delete the target movie clip
    removeMovieClip(this);
    };
}
var jpeg_mcl:MovieClipLoader = new MovieClipLoader();
jpeg_mcl.addListener(listenerObject);
jpeg_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    this.createEmptyMovieClip("jpeg_mc", this.getNextHighestDepth()));
```

If you work with `setInterval()` within classes, you need to be sure to use the `this` keyword when you call the function. The `setInterval()` function does not have access to class members if you do not use the keyword. This is illustrated in the following example. For a FLA file with a button called `deleteUser_btn`, add the following ActionScript to Frame 1:

```
var me:User = new User("Gary");
this.deleteUser_btn.onRelease = function() {
    trace("Goodbye, "+me.username);
    clearInterval(me.intervalID);
    delete me;
};
```

Then create a file called User.as in the same directory as your FLA file. Enter the following ActionScript:

```
class User {
    var intervalID:Number;
    var username:String;
    function User(param_username:String) {
    trace("Welcome, "+param_username);
    this.username = param_username;
    this.intervalID = setInterval(this, "traceUsername", 1000, this.username);
    }
    function traceUsername(str:String) {
    trace(this.username+" is "+getTimer()/1000+" seconds old, happy birthday.");
    }
}
```

See also

clearInterval function

setProperty function

`setProperty(target:Object, property:Object, expression:Object) : Void`

Changes a property value of a movie clip as the movie clip plays.

Availability

Flash Lite 1.0

Parameters

`target:Object` - The path to the instance name of the movie clip whose property is to be set.

`property:Object` - The property to be set.

`expression:Object` - Either the new literal value of the property, or an equation that evaluates to the new value of the property.

Example

The following ActionScript creates a new movie clip and loads an image into it. The `_x`and `_y` coordinates are set for the clip using `setProperty()`. When you click the button called `right_btn`, the `_x` coordinate of a movie clip named `params_mc` is incremented by 20 pixels.

```
this.createEmptyMovieClip("params_mc", 999);
params_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
setProperty(this.params_mc, _y, 20);
setProperty(this.params_mc, _x, 20);
this.right_btn.onRelease = function() {
    setProperty(params_mc, _x, getProperty(params_mc, _x)+20);
};
```

See also

getProperty function

startDrag function

`startDrag(target:Object [, lock:Boolean, left:Number, top:Number, right:Number, bottom:Number]) : Void`

Makes the *target* movie clip draggable while the movie plays. Only one movie clip can be dragged at a time. After a `startDrag()` operation is executed, the movie clip remains draggable until it is explicitly stopped by `stopDrag()` or until a `startDrag()` action for another movie clip is called.

Note: This method is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Parameters

`target:Object` - The target path of the movie clip to drag.

`lock:Boolean` [optional] - A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (`true`) or locked to the point where the user first clicked the movie clip (`false`).

`left,top,right,bottom:Number` [optional] - Values relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip.

Example

The following example creates a movie clip, `pic_mc`, at runtime that users can drag to any location by attaching the `startDrag()` and `stopDrag()` actions to the movie clip. An image is loaded into `pic_mc` using the MovieClipLoader class.

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();
pic_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    this.createEmptyMovieClip("pic_mc", this.getNextHighestDepth()));
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc) {
    target_mc.onPress = function() {
    startDrag(this);
    };
    target_mc.onRelease = function() {
    stopDrag();
    };
};
pic_mcl.addListener(listenerObject);
```

See also

`stopDrag function`, `_droptarget (MovieClip._droptarget property)`, `startDrag (MovieClip.startDrag method)`

stop function

`stop() : Void`

Stops the SWF file that is currently playing. The most common use of this action is to control movie clips with buttons.

Availability

Flash Lite 1.0

See also

`gotoAndStop function`, `gotoAndStop (MovieClip.gotoAndStop method)`

stopAllSounds function

```
stopAllSounds() : Void
```

Stops all sounds currently playing in a SWF file without stopping the playhead. Sounds set to stream will resume playing as the playhead moves over the frames in which they are located.

Availability

Flash Lite 1.0

Example

The following code creates a text field, in which the song's ID3 information appears. A new Sound object instance is created, and your MP3 is loaded into the SWF file. ID3 information is extracted from the sound file. When the user clicks `stop_mc`, the sound is paused. When the user clicks `play_mc`, the song resumes from its paused position.

```
this.createTextField("songinfo_txt", this.getNextHighestDepth, 0, 0, Stage.width, 22);
var bg_sound:Sound = new Sound();
bg_sound.loadSound("yourSong.mp3", true);
bg_sound.onID3 = function() {
    songinfo_txt.text = "(" + this.id3.artist + ") " + this.id3.album + " - " + this.id3.track
+ " - "
    + this.id3.songname;
    for (prop in this.id3) {
    trace(prop+" = "+this.id3[prop]);
    }
    trace("ID3 loaded.");
};
this.play_mc.onRelease = function() {
    /* get the current offset. if you stop all sounds and click the play button, the MP3
continues from
    where it was stopped, instead of restarting from the beginning. */
    var numSecondsOffset:Number = (bg_sound.position/1000);
    bg_sound.start(numSecondsOffset);
};
this.stop_mc.onRelease = function() {
    stopAllSounds();
};
```

See also

Sound

stopDrag function

```
stopDrag() : Void
```

Stops the current drag operation.

Note: This method is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following code, placed in the main Timeline, stops the drag action on the movie clip instance `my_mc` when the user releases the mouse button:

```
my_mc.onPress = function () {
    startDrag(this);
}

my_mc.onRelease = function() {
    stopDrag();
}
```

See also

`startDrag function`, `_droptarget (MovieClip._droptarget property)`, `startDrag (MovieClip.startDrag method)` `stopDrag (MovieClip.stopDrag method)`

String function

```
String(expression:Object) : String
```

Returns a string representation of the specified parameter, as described in the following list:

- If *expression* is a number, the return string is a text representation of the number.
- If *expression* is a string, the return string is *expression*.
- If *expression* is an object, the return value is a string representation of the object generated by calling the string property for the object or by calling `Object.toString()` if no such property exists.
- If *expression* is a Boolean value, the return string is `"true"` or `"false"`.
- If *expression* is a movie clip, the return value is the target path of the movie clip in slash (/) notation.

If *expression* is `undefined`, the return values are as follows:

- In files published for Flash Player 6 or earlier, the result is an empty string ("").
- In files published for Flash Player 7 or later, the result is `undefined`.

Note: *Slash notation is not supported by ActionScript 2.0.*

Availability

Flash Lite 1.0

Parameters

`expression:Object` - An expression to convert to a string.

Returns

`String` - A string.

Example

In the following example, you use ActionScript to convert specified expressions to a string:

```
var string1:String = String("3");
var string2:String = String("9");
trace(string1+string2); // output: 39
```

Because both parameters are strings, the values are concatenated rather than added.

See also

`toString (Number.toString method)`, `toString (Object.toString method)`, `String`, `" string delimiter operator`

substring function

```
substring("string", index, count) : String
```

Deprecated since Flash Player 5. This function was deprecated in favor of `String.substr()`.

Extracts part of a string. This function is one-based, whereas the String object methods are zero-based.

Availability

Flash Lite 1.0

Parameters

`string:String` - The string from which to extract the new string.

`index:Number` - The number of the first character to extract.

`count:Number` - The number of characters to include in the extracted string, not including the index character.

Returns

`String` - The extracted substring.

See also

`substr (String.substr method)`

targetPath function

```
targetpath(targetObject:Object) : String
```

Returns a string containing the target path of a MovieClip, Button, or TextField object. The target path is returned in dot (.) notation. To retrieve the target path in slash (/) notation, use the `_target` property.

Availability

Flash Lite 2.0

Parameters

`targetObject:Object` - Reference (for example, `_root` or `_parent`) to the object for which the target path is being retrieved. This can be a MovieClip, Button, or TextField object.

Returns

`String` - A string containing the target path of the specified object.

Example

The following example traces the target path of a movie clip as soon as it loads:

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());
trace(targetPath(myClip_mc)); // _level0.myClip_mc
```

See also

`eval function`

tellTarget function

```
tellTarget("target") { statement(s); }
```

Deprecated since Flash Player 5. Adobe recommends that you use dot (.) notation and the `with` statement.

Applies the instructions specified in the *statements* parameter to the Timeline specified in the *target* parameter. The `tellTarget` action is useful for navigation controls. Assign `tellTarget` to buttons that stop or start movie clips elsewhere on the Stage. You can also make movie clips go to a particular frame in that clip. For example, you might assign `tellTarget` to buttons that stop or start movie clips on the Stage or prompt movie clips to jump to a particular frame.

In Flash 5 or later, you can use dot (.) notation instead of the `tellTarget` action. You can use the `with` action to issue multiple actions to the same Timeline. You can use the `with` action to target any object, whereas the `tellTarget` action can target only movie clips.

Availability

Flash Lite 1.0

Parameters

`target:String` - A string that specifies the target path of the Timeline to be controlled.

`statement(s):Object` - The instructions to execute if the condition is `true`.

Example

This tellTarget statement controls the movie clip instance ball on the main Timeline. Frame 1 of the ball instance is blank and has a stop() action so it isn't visible on the Stage. When you click the button with the following action, tellTarget tells the playhead in ball to go to Frame 2, where the animation starts:

```
on(release) {
tellTarget("_parent.ball") {
gotoAndPlay(2);
}
}
```

The following example uses dot (.) notation to achieve the same results:

```
on(release) {
_parent.ball.gotoAndPlay(2);
}
```

If you need to issue multiple commands to the ball instance, you can use the `with` action, as shown in the following statement:

```
on(release) {
with(_parent.ball) {
gotoAndPlay(2);
_alpha = 15;
_xscale = 50;
_yscale = 50;
}
}
```

See also

`with statement`

toggleHighQuality function

`toggleHighQuality()`

Deprecated since Flash Player 5. This function was deprecated in favor of `_quality`.

Turns anti-aliasing on and off in Flash Player. Anti-aliasing smooths the edges of objects and slows down SWF playback. This action affects all SWF files in Flash Player.

Availability

Flash Lite 1.0

Example

The following code could be applied to a button that, when clicked, would toggle anti-aliasing on and off:

```
on(release) {
toggleHighQuality();
}
```

See also

`_highquality property`, `_quality property`

trace function

`trace(expression:Object)`

You can use Flash Debug Player to capture output from the `trace()` function and write that output to the log file.

Statement; evaluates the expression and displays the result in the Output panel in test mode.

Use this statement to record programming notes or to display messages in the Output panel while testing a SWF file. Use the *expression* parameter to check whether a condition exists, or to display values in the Output panel. The `trace()` statement is similar to the `alert` function in JavaScript.

You can use the Omit Trace Actions command in the Publish Settings dialog box to remove `trace()` actions from the exported SWF file.

Availability

Flash Lite 1.0

Parameters

`expression:Object` - An expression to evaluate. When a SWF file is opened in the Flash authoring tool (using the Test Movie command), the value of the *expression* parameter is displayed in the Output panel.

Example

The following example uses a `trace()` statement to display in the Output panel the methods and properties of the dynamically created text field called `error_txt`:

```
this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
for (var i in error_txt) {
trace("error_txt."+i+" = "+error_txt[i]);
}
/* output:
error_txt.styleSheet = undefined
error_txt.mouseWheelEnabled = true
error_txt.condenseWhite = false
...
error_txt.maxscroll = 1
error_txt.scroll = 1
*/
```

unescape function

```
unescape(x:String) : String
```

Evaluates the parameter *x* as a string, decodes the string from URL-encoded format (converting all hexadecimal sequences to ASCII characters), and returns the string.

Availability

Flash Lite 1.1

Parameters

`string:String` - A string with hexadecimal sequences to escape.

Returns

`String` - A string decoded from a URL-encoded parameter.

Example

The following example shows the escape-to-unescape conversion process:

```
var email:String = "user@somedomain.com";
trace(email);
var escapedEmail:String = escape(email);
trace(escapedEmail);
var unescapedEmail:String = unescape(escapedEmail);
trace(unescapedEmail);
```

The following result is displayed in the Output panel.

```
user@somedomain.com
user%40somedomain%2Ecom
user@somedomain.com
```

unloadMovie function

```
unloadMovie(target:MovieClip) : Void
unloadMovie(target:String) : Void
```

Removes a movie clip that was loaded by means of `loadMovie()` from Flash Player. To unload a movie clip that was loaded by means of `loadMovieNum()`, use `unloadMovieNum()` instead of `unloadMovie()`.

Availability

Flash Lite 1.1

Parameters

`target` - The target path of a movie clip. This parameter can be either a string (e.g. "my_mc") or a direct reference to the movie clip instance (e.g. my_mc). Parameters that can accept more than one data type are listed as type `Object`.

Example

The following example creates a new movie clip called `pic_mc` and loads an image into that clip. It is loaded using the MovieClipLoader class. When you click the image, the movie clip unloads from the SWF file:

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();
pic_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    this.createEmptyMovieClip("pic_mc", this.getNextHighestDepth()));
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc) {
    target_mc.onRelease = function() {
    unloadMovie(pic_mc);
    /* or you could use the following, which refers to the movie clip referenced by 'target_mc'.
*/
    //unloadMovie(this);
    };
};
pic_mcl.addListener(listenerObject);
```

See also

`loadMovie (MovieClip.loadMovie method)`, `unloadClip (MovieClipLoader.unloadClip method)`

unloadMovieNum function

```
unloadMovieNum(level:Number) : Void
```

Removes a SWF or image that was loaded by means of loadMovieNum() from Flash Player. To unload a SWF or image that was loaded with `MovieClip.loadMovie()`, use `unloadMovie()` instead of `unloadMovieNum()`.

Availability

Flash Lite 1.1

Parameters

`level:Number` - The level (`_level` *N*) of a loaded movie.

Example

The following example loads an image into a SWF file. When you click `unload_btn`, the loaded content is removed.

```
loadMovieNum("yourimage.jpg", 1);
unload_btn.onRelease = function() {
    unloadMovieNum(1);
}
```

See also

`loadMovieNum function`, `unloadMovie function`, `loadMovie (MovieClip.loadMovie method)`

Global properties

Global properties are available in every script, and are visible to every Timeline and scope in your document. For example, global properties allow access to the timelines of other loaded movie clips, both relative (`_parent`) and absolute (`_root`). They also let you restrict (`this`) or expand (`super`) scope. And, you can use global properties to adjust runtime settings like screen reader accessibility, playback quality, and sound buffer size.

Global properties summary

Modifiers	Property	Description
	$version	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.version` property. Contains the version number of Flash Lite.
	_cap4WayKeyAS	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.has4WayKeyAS` property. Indicates whether Flash Lite executes ActionScript expressions attached to key event handlers associated with the Right, Left, Up, and Down Arrow keys.
	_capCompoundSound	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasCompoundSound` property. Indicates whether Flash Lite can process compound sound data.
	_capEmail	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasEmail` property. Indicates whether the Flash Lite client can send e-mail messages by using the `GetURL()` ActionScript command.
	_capLoadData	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasDataLoading` property. Indicates whether the host application can dynamically load additional data through calls to the `loadMovie()`, `loadMovieNum()`, `loadVariables()`, and `loadVariablesNum()` functions.

Modifiers	Property	Description
	_capMFi	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMFi` property. Indicates whether the device can play sound data in the Melody Format for i-mode (MFi) audio format.
	_capMIDI	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMIDI` property. Indicates whether the device can play sound data in the Musical Instrument Digital Interface (MIDI) audio format.
	_capMMS	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMMS` property. Indicates whether Flash Lite can send Multimedia Messaging Service (MMS) messages by using the `GetURL()` ActionScript command. If so, this variable is defined and has a value of 1; if not, this variable is undefined.
	_capSMAF	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasSMAF` property. Indicates whether the device can play multimedia files in the Synthetic music Mobile Application Format (SMAF). If so, this variable is defined and has a value of 1; if not, this variable is undefined.
	_capSMS	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasSMS` property. Indicates whether Flash Lite can send Short Message Service (SMS) messages by using the `GetURL()` ActionScript command.
	_capStreamSound	**Deprecated** since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasStreamingAudio` property. Indicates whether the device can play streaming (synchronized) sound.
	_focusrect	Property (global); specifies whether a yellow rectangle appears around the button or movie clip that has keyboard focus.
	_forceframerate	Tells the Flash Lite player to render at the specified frame rate.
	_global	A reference to the global object that holds the core ActionScript classes, such as String, Object, Math, and Array.
	_highquality	**Deprecated** since Flash Player 5. This property was deprecated in favor of `_quality`. Specifies the level of anti-aliasing applied to the current SWF file.
	_level	A reference to the root Timeline of `_level`*N*.

Modifiers	Property	Description
	maxscroll	**Deprecated** since Flash Player 5. This property was deprecated in favor of `TextField.maxscroll`. Indicates the line number of the top line of visible text in a text field when the bottom line in the field is also visible.
	_parent	Specifies or returns a reference to the movie clip or object that contains the current movie clip or object.
	_quality	Sets or retrieves the rendering quality used for a movie clip.
	_root	Specifies or returns a reference to the root movie clip Timeline.
	scroll	**Deprecated** since Flash Player 5. This property was deprecated in favor of `TextField.scroll`. Controls the display of information in a text field associated with a variable.
	_soundbuftime	Establishes the number of seconds of streaming sound to buffer.
	this	References an object or movie clip instance.

$version property

`$version`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.version` property.

String variable; contains the version number of Flash Lite. It contains a major number, minor number, build number, and an internal build number, which is generally `0` in all released versions. The major number reported for all Flash Lite 1.x products is 5. Flash Lite 1.0 has a minor number of 1; Flash Lite 1.1 has a minor number of 2.

Availability

Flash Lite 1.1

Example

In the Flash Lite 1.1 player, the following code sets the value of `myVersion` to "5, 2, 12, 0":

```
myVersion = $version;
```

See also

`version (capabilities.version property)`

_cap4WayKeyAS property

`_cap4WayKeyAS`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.has4WayKeyAS` property.

Numeric variable; indicates whether Flash Lite executes ActionScript expressions attached to key event handlers associated with the Right, Left, Up, and Down Arrow keys. This variable is defined and has a value of 1 only when the host application uses four-way key navigation mode to move between Flash controls (buttons and input text fields). Otherwise, this variable is undefined.

When one of the four-way keys is pressed, if the value of the `_cap4WayKeyAS` variable is `1`, Flash Lite first looks for a handler for that key. If it finds none, Flash control navigation occurs. However, if an event handler is found, no navigation action occurs for that key. For example, if a key press handler for the Down Arrow key is found, the user cannot navigate.

Availability

Flash Lite 1.1

Example

The following example sets `canUse4Way` to `1` in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 (however, not all Flash Lite 1.1 phones support four-way keys, so this code is still dependent on the phone):

```
canUse4Way = _cap4WayKeyAS;
    if (canUse4Way == 1) {
    msg = "Use your directional joypad to navigate this application";
    } else {
    msg = "Please use the 2 key to scroll up, the 6 key to scroll right,

the 8 key to scroll down, and the 4 key to scroll left.";
    }
```

See also

capabilities (System.capabilities)

_capCompoundSound property

`_capCompoundSound`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasCompoundSound` property.

Numeric variable; indicates whether Flash Lite can process compound sound data. If so, this variable is defined and has a value of 1; if not, this variable is undefined. For example, a single Flash file can contain the same sound represented in both MIDI and MFi formats. The player will then play back data in the appropriate format based on the format supported by the device. This variable defines whether the Flash Lite player supports this ability on the current handset.

Availability

Flash Lite 1.1

Example

In the following example, `useCompoundSound` is set to `1` in Flash Lite 1.1, but is undefined in Flash Lite 1.0:

```
useCompoundSound = _capCompoundSound;

if (useCompoundSound == 1) {
    gotoAndPlay("withSound");
} else {
    gotoAndPlay("withoutSound");
```

See also

capabilities (System.capabilities)

_capEmail property

`_capEmail`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasEmail` property.

Numeric variable; indicates whether the Flash Lite client can send e-mail messages by using the `GetURL()` ActionScript command. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Availability

Flash Lite 1.1

Example

If the host application can send e-mail messages by using the `GetURL()` ActionScript command, the following example sets `canEmail()` to `1`:

```
canEmail = _capEmail;

if (canEmail == 1) {
    getURL("mailto:someone@somewhere.com?subject=foo&body=bar");
}
```

See also

`capabilities (System.capabilities)`

_capLoadData property

`_capLoadData`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasDataLoading` property.

Numeric variable; indicates whether the host application can dynamically load additional data through calls to the `loadMovie()`, `loadMovieNum()`, `loadVariables()`, and `loadVariablesNum()` functions. If so, this variable is defined and has a value of `1`; if not, this variable is undefined.

Availability

Flash Lite 1.1

Example

If the host application can perform dynamic loading of movies and variables, the following example sets `CanLoad` to `1`:

```
canLoad = _capLoadData;

if (canLoad == 1) {
    loadVariables("http://www.somewhere.com/myVars.php", GET);
} else {
    trace ("client does not support loading dynamic data");
}
```

See also

`capabilities (System.capabilities)`

_capMFi property

`_capMFi`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMFi` property.

Numeric variable; indicates whether the device can play sound data in the Melody Format for i-mode (MFi) audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Availability

Flash Lite 1.1

Example

If the device can play MFi sound data, the following example sets `canMFi` to `1`:

```
canMFi = _capMFi;

if (canMFi == 1) {
    // send movieclip buttons to frame with buttons that trigger events

sounds
    tellTarget("buttons") {
    gotoAndPlay(2);
    }
}
```

See also

`hasMFI (capabilities.hasMFI property)`

_capMIDI property

`_capMIDI`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMIDI` property.

Numeric variable; indicates whether the device can play sound data in the Musical Instrument Digital Interface (MIDI) audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Availability

Flash Lite 1.1

Example

If the device can play MIDI sound data, the following example sets `_capMIDI` to `1`:

```
canMIDI = _capMIDI;

if (canMIDI == 1) {
    // send movieclip buttons to frame with buttons that trigger events

sounds
    tellTarget("buttons") {
    gotoAndPlay(2);
    }
}
```

See also

capabilities (System.capabilities)

_capMMS property

_capMMS

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the System.capabilities.hasMMS property.

Numeric variable; indicates whether Flash Lite can send Multimedia Messaging Service (MMS) messages by using the GetURL() ActionScript command. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Availability

Flash Lite 1.1

Example

The following example sets canMMS to 1 in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 (however, not all Flash Lite 1.1 phones can send MMS messages, so this code is still dependent on the phone):

```
on(release) {
    canMMS = _capMMS;
    if (canMMS == 1) {
    // send an MMS
    myMessage = "mms:4156095555?body=sample mms message";
    } else {
    // send an SMS
    myMessage = "sms:4156095555?body=sample sms message";
    }
    getURL(myMessage);
}
```

See also

capabilities (System.capabilities)

_capSMAF property

_capSMAF

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the System.capabilities.hasSMAF property.

Numeric variable; indicates whether the device can play multimedia files in the Synthetic music Mobile Application Format (SMAF). If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Availability
Flash Lite 1.1

Example
The following example sets `canSMAF` to `1` in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 (however, not all Flash Lite 1.1 phones can send SMAF messages, so this code is still dependent on the phone):

```
canSMAF = _capSMAF;

if (canSMAF) {
    // send movieclip buttons to frame with buttons that trigger events

sounds
    tellTarget("buttons") {
    gotoAndPlay(2);
    }
}
```

See also
`capabilities (System.capabilities)`

_capSMS property

`_capSMS`

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasSMS` property.

Numeric variable; indicates whether Flash Lite can send Short Message Service (SMS) messages by using the `GetURL()` ActionScript command. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Availability
Flash Lite 1.1

Example
The following example sets `canSMS` to `1` in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 (however, not all Flash Lite 1.1 phones can send SMS messages, so this code is still dependent on the phone):

```
on(release) {
    canSMS = _capSMS;
    if (canSMS) {
    // send an SMS
    myMessage = "sms:4156095555?body=sample sms message";
    getURL(myMessage);
    }
}
```

See also
`capabilities (System.capabilities)`

_capStreamSound property

```
_capStreamSound
```

Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasStreamingAudio` property.

Numeric variable; indicates whether the device can play streaming (synchronized) sound. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Availability

Flash Lite 1.1

Example

The following example plays streaming sound if `canStreamSound` is enabled:

```
on(press) {
    canStreamSound = _capStreamSound;
    if (canStreamSound) {
    // play a streaming sound in a movieclip with this button
    tellTarget("music") {
    gotoAndPlay(2);
    }
    }
    }
```

See also

`capabilities (System.capabilities)`

_focusrect property

```
_focusrect = Boolean;
```

Specifies whether a yellow rectangle appears around the button or movie clip that has keyboard focus. If `_focusrect` is set to its default value of `true`, a yellow rectangle appears around the currently focused button or movie clip as the user presses the Tab key to navigate through objects in a SWF file. Specify `false` if you do not want to show the yellow rectangle. This is a property that can be overridden for specific instances.

If the global `_focusrect` property is set to `false`, the default behavior for all buttons and movie clips is that keyboard navigation is limited to the Tab key. All other keys, including the Enter and arrow keys, are ignored. To restore full keyboard navigation, you must set `_focusrect` to `true`. To restore full keyboard functionality for a specific button or movie clip, you can override this global property by using either `Button._focusrect` or `MovieClip._focusrect`.

Note: If you use a component, FocusManager overrides Flash Player's focus handling, including use of this global property.

Note: For the Flash Lite 2.0 player, when the `_focusrect` property is disabled (such as `Button.focusRect = false` or `MovieClip.focusRect = false`), the button or movie clip still receives all events. This behavior is different from the Flash player, for when the `_focusrect` property is disabled, the button or movie clip will receive the `rollOver` and `rollOut` events but will not receive the `press` and `release` events.

Also for Flash Lite 2.0, you can change the color of the focus rectangle by using the fscommand2 `SetFocusRectColor` command. This behavior is different from Flash Player, where the color of the focus rectangle is restricted to yellow.

Availability
Flash Lite 1.0

Example
The following example demonstrates how to hide the yellow rectangle around any instances in a SWF file when they have focus in a browser window. Create some buttons or movie clips and add the following ActionScript in Frame 1 of the Timeline:

```
_focusrect = false;
```

See also
`_focusrect (Button._focusrect property)`, `_focusrect (MovieClip._focusrect property)`

_forceframerate property

`_forceframerate`

If set to `true`, this property tells the Flash Lite player to render at the specified frame rate. You can use this property for pseudo-synchronized sound when the content contains device sound. It is set to `false` by default, which causes Flash Lite to render normally. When set to `true`, the Flash Lite player might skip rendering certain frames to maintain the frame rate.

Availability
Flash Lite 2.0

_global property

`_global.identifier`

A reference to the global object that holds the core ActionScript classes, such as String, Object, Math, and Array. For example, you could create a library that is exposed as a global ActionScript object, similar to the Math or Date object. Unlike Timeline-declared or locally declared variables and functions, global variables and functions are visible to every timeline and scope in the SWF file, provided they are not obscured by identifiers with the same names in inner scopes.

Note: When setting the value of a global variable, you must use the fully qualified name of the variable, for instance, _global.variableName. Failure to do so creates a local variable of the same name that obscures the global variable you are attempting to set.

Availability
Flash Lite 2.0

Returns
A reference to the global object that holds the core ActionScript classes, such as String, Object, Math, and Array.

Example
The following example creates a top-level function, `factorial()`, that is available to every timeline and scope in a SWF file:

```
_global.factorial = function(n:Number) {
    if (n&lt;=1) {
    return 1;
    } else {
    return n*factorial(n-1);
    }
}
// Note: factorial 4 == 4*3*2*1 == 24
trace(factorial(4)); // output: 24
```

The following example shows how the failure to use the fully qualified variable name when setting the value of a global variable leads to unexpected results:

```
_global.myVar = "global";
trace("_global.myVar: " + _global.myVar); // _global.myVar: global
trace("myVar: " + myVar); // myVar: global

myVar = "local";
trace("_global.myVar: " + _global.myVar); // _global.myVar: global
trace("myVar: " + myVar); // myVar: local
```

See also

`set variable statement`

_highquality property

`_highquality`

Deprecated since Flash Player 5. This property was deprecated in favor of `_quality`.

Specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply the best quality. Specify 1 (high quality) to apply anti-aliasing. Specify 0 (low quality) to prevent anti-aliasing.

Availability

Flash Lite 1.0

Example

The following ActionScript is placed on the main timeline, and sets the global quality property to apply anti-aliasing.

```
_highquality = 1;
```

See also

`_quality property`

_level property

`_levelN`

A reference to the root Timeline of `_levelN`. You must use `loadMovieNum()` to load SWF files into the Flash Player before you use the `_level` property to target them. You can also use `_levelN` to target a loaded SWF file at the level assigned by *N*.

The initial SWF file loaded into an instance of the Flash Player is automatically loaded into `_level0`. The SWF file in `_level0` sets the frame rate, background color, and frame size for all subsequently loaded SWF files. SWF files are then stacked in higher-numbered levels above the SWF file in `_level0`.

You must assign a level to each SWF file that you load into the Flash Player using `loadMovieNum()`. You can assign levels in any order. If you assign a level that already contains a SWF file (including `_level0`), the SWF file at that level is unloaded and replaced by the new SWF file.

Availability
Flash Lite 1.0

Example
The following example stops the playhead in the main timeline of the SWF file sub.swf that is loaded into `_level9`. The sub.swf file contains animation and is in the same directory as the document that contains the following ActionScript:

```
loadMovieNum("sub.swf", 9);
myBtn_btn.onRelease = function() {
    _level9.stop();
};
```

You could replace `_level9.stop()` in the previous example with the following code:

```
_level9.gotoAndStop(5);
```

This action sends the playhead in the main Timeline of the SWF file in `_level9` to Frame 5 instead of stopping the playhead.

See also
`loadMovie function`, `swapDepths (MovieClip.swapDepths method)`

maxscroll property

variable_name`.maxscroll`

Deprecated since Flash Player 5. This property was deprecated in favor of `TextField.maxscroll`.

Indicates the line number of the top line of visible text in a text field when the bottom line in the field is also visible. The `maxscroll` property works with the `scroll` property to control how information appears in a text field. This property can be retrieved, but not modified.

Availability
Flash Lite 1.1

See also
`maxscroll (TextField.maxscroll property)`, `scroll (TextField.scroll property)`

_parent property

`_parent.`*property*
`_parent._parent.`*property*

Specifies or returns a reference to the movie clip or object that contains the current movie clip or object. The current object is the object containing the ActionScript code that references `_parent`. Use `_parent` to specify a relative path to movie clips or objects that are above the current movie clip or object.

Availability
Flash Lite 2.0

Example
In the following example, there is a movie clip on the Stage with the instance name `square_mc`. Within that movie clip is another movie clip with an instance name `circle_mc`. The following ActionScript lets you modify the `circle_mc` parent instance (which is `square_mc`) when the circle is clicked. When you are working with relative addressing (using `_parent` instead of `_root`), it might be easier to use the Insert Target Path button in the Actions panel at first.

```
this.square_mc.circle_mc.onRelease = function() {
    this._parent._alpha -= 5;
};
```

See also
`_root property`, `targetPath function`

_quality property

`_quality:String`

Sets or retrieves the rendering quality used for a movie clip. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

Value	Description	Graphic Anti-Aliasing	Bitmap Smoothing
`"LOW"`	Low rendering quality.	Graphics are not anti-aliased.	Bitmaps are not smoothed.
`"MEDIUM"`	Medium rendering quality. This setting is suitable for movies that do not contain text.	Graphics are anti-aliased using a 2 x 2 pixel grid.	Bitmaps are not smoothed.
`"HIGH"`	High rendering quality. This setting is the default rendering quality setting that Flash uses.	Graphics are anti-aliased using a 4 x 4 pixel grid.	Bitmaps are not smoothed.

Availability
Flash Lite 2.0

Example
The following example sets the rendering quality to `LOW`:

```
_quality = "LOW";
```

_root property

```
_root.movieClip
_root.action
_root.property
```

Specifies or returns a reference to the root movie clip Timeline. If a movie clip has multiple levels, the root movie clip Timeline is on the level containing the currently executing script. For example, if a script in level 1 evaluates `_root`, `_level1` is returned.

Specifying `_root` is the same as using the deprecated slash notation (`/`) to specify an absolute path within the current level.

Note: If a movie clip that contains _root is loaded into another movie clip, _root refers to the Timeline of the loading movie clip, not the Timeline that contains _root. If you want to ensure that _root refers to the Timeline of the loaded movie clip even if it is loaded into another movie clip, use MovieClip._lockroot.

Availability
Flash Lite 2.0

Parameters
movieClip : `String` - The instance name of a movie clip.

action : `String` - An action or field.

property : `String` - A property of the MovieClip object.

Example
The following example stops the Timeline of the level containing the currently executing script:

```
_root.stop();
```

The following example traces variables and instances in the scope of `_root`:

```
for (prop in _root) {
    trace("_root."+prop+" = "+_root[prop]);
}
```

See also
`_lockroot (MovieClip._lockroot property)`, `_parent property`, `targetPath function`

scroll property

`textFieldVariableName``.scroll = x`

Deprecated since Flash Player 5. This property was deprecated in favor of `TextField.scroll`.

Controls the display of information in a text field associated with a variable. The `scroll` property defines where the text field begins displaying content; after you set it, Flash Player updates it as the user scrolls through the text field. The `scroll` property is useful for directing users to a specific paragraph in a long passage or creating scrolling text fields. This property can be retrieved and modified.

Availability
Flash Lite 1.1

Example
The following code is attached to an Up button that scrolls the text field named myText:

```
on (release) {
myText.scroll = myText.scroll + 1;
}
```

See also

maxscroll (TextField.maxscroll property), scroll (TextField.scroll property)

_soundbuftime property

```
_soundbuftime:Number = integer
```

Establishes the number of seconds of streaming sound to buffer. The default value is 5 seconds.

Availability

Flash Lite 2.0

Parameters

integer : `Number` - The number of seconds before the SWF file starts to stream.

Example

The following example streams an MP3 file and buffers the sound before it plays for the user. Two text fields are created at runtime to hold a timer and debugging information. The `_soundbuftime` property is set to buffer the MP3 for 10 seconds. A new Sound object instance is created for the MP3.

```
// create text fields to hold debug information.
this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
this.createTextField("debug_txt", this.getNextHighestDepth(), 0, 20, 100, 22);
// set the sound buffer to 10 seconds.
_soundbuftime = 10;
// create the new sound object instance.
var bg_sound:Sound = new Sound();
// load the MP3 sound file and set streaming to true.
bg_sound.loadSound("yourSound.mp3", true);
// function is triggered when the song finishes loading.
bg_sound.onLoad = function() {
    debug_txt.text = "sound loaded";
};
debug_txt.text = "sound init";
function updateCounter() {
    counter_txt.text++;
}
counter_txt.text = 0;
setInterval(updateCounter, 1000);
```

this property

```
this
```

References an object or movie clip instance. When a script executes, `this` references the movie clip instance that contains the script. When a field is called, `this` contains a reference to the object that contains the called field.

Inside an `on()` event handler attached to a button, `this` refers to the Timeline that contains the button. Inside an `onClipEvent()` event handler attached to a movie clip, `this` refers to the Timeline of the movie clip itself.

Because `this` is evaluated in the context of the script that contains it, you can't use `this` in a script to refer to a variable defined in a class file. Create ApplyThis.as, and enter the following code:

```
class ApplyThis {
var str:String = "Defined in ApplyThis.as";
function conctStr(x:String):String {
return x+x;
}
function addStr():String {
return str;
}
}
```

Then, in a FLA or AS file, add the following ActionScript:

```
var obj:ApplyThis = new ApplyThis();
var abj:ApplyThis = new ApplyThis();
abj.str = "defined in FLA or AS";
trace(obj.addStr.call(abj, null)); //output: defined in FLA or AS
trace(obj.addStr.call(this, null)); //output: undefined
trace(obj.addStr.call(obj, null)); //output: Defined in applyThis.as
```

Similarly, to call a function defined in a dynamic class, you must use `this` to invoke the function in the proper scope:

```
// incorrect version of Simple.as
/*
dynamic class Simple {
function callfunc() {
trace(func());
}
}
*/
// correct version of Simple.as
dynamic class simple {
function callfunc() {
trace(this.func());
}
}
```

Inside the FLA or AS file, add the following ActionScript:

```
var obj:Simple = new Simple();
obj.num = 0;
obj.func = function() {
return true;
};
obj.callfunc();
// output: true
```

You get a syntax error when you use the incorrect version of Simple.as.

Availability

Flash Lite 2.0

Example

In the following example, the keyword `this` references the Circle object:

```
function Circle(radius:Number):Void {
    this.radius = radius;
    this.area = Math.PI*Math.pow(radius, 2);
}
var myCircle = new Circle(4);
trace(myCircle.area);
```

In the following statement assigned to a frame inside a movie clip, the keyword `this` references the current movie clip.

```
// sets the alpha property of the current movie clip to 20
this._alpha = 20;
```

In the following statement inside a MovieClip.onPress handler, the keyword `this` references the current movie clip:

```
this.square_mc.onPress = function() {
    startDrag(this);
};
this.square_mc.onRelease = function() {
    stopDrag();
};
```

See also

Constants, onClipEvent handler

Operators

Symbolic operators are characters that specify how to combine, compare, or modify the values of an expression.

Operators summary

Operator	Description
+ (addition)	Adds numeric expressions or concatenates (combines) strings.
+= (addition assignment)	Assigns *expression1* the value of ***expression1 + expression2.***
[] (array access)	Initializes a new array or multidimensional array with the specified elements (*a0*, and so on), or accesses elements in an array.
= (assignment)	Assigns the value of ***expression2*** (the parameter on the right) to the variable, array element, or property in *expression1*.
& (bitwise AND)	Converts ***expression1 and expression2*** to 32-bit unsigned integers, and performs a Boolean AND operation on each bit of the integer parameters.
&= (bitwise AND assignment)	Assigns ***expression1*** the value of ***expression1& expression2***.
<< (bitwise left shift)	Converts ***expression1*** and ***expression2*** to 32-bit integers, and shifts all the bits in ***expression1*** to the left by the number of places specified by the integer resulting from the conversion of ***expression2***.
<<= (bitwise left shift and assignment)	This operator performs a bitwise left shift (<<) operation and stores the contents as a result in ***expression1***.
~ (bitwise NOT)	Also known as the one's complement operator or the bitwise complement operator.

Operator	Description
`|` (bitwise OR)	Converts ***expression1*** and ***expression2*** to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits of either ***expression1*** or ***expression2*** are 1.
`|=` (bitwise OR assignment)	Assigns *expression1* the value of *expression1* | *expression2*.
`>>` (bitwise right shift)	Converts *expression1* and *expression2* to 32-bit integers, and shifts all the bits in *expression1* to the right by the number of places specified by the integer that results from the conversion of *expression2*.
`>>=` (bitwise right shift and assignment)	This operator performs a bitwise right-shift operation and stores the contents as a result in *expression1*.
`>>>` (bitwise unsigned right shift)	The same as the bitwise right shift (`>>`) operator except that it does not preserve the sign of the original *expression* because the bits on the left are always filled with 0. Floating-point numbers are converted to integers by discarding any digits after the decimal point.
`>>>=` (bitwise unsigned right shift and assignment)	Performs an unsigned bitwise right-shift operation and stores the contents as a result in *expression1*.
`^` (bitwise XOR)	Converts *expression1* and *expression2* to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits in *expression1* or *expression2*, but not both, are 1.
`^=` (bitwise XOR assignment)	Assigns ***expression1*** the value of ***expression1 ^ expression2***.
`/*` (block comment delimiter)	Indicates one or more lines of script comments.
`,` (comma)	Evaluates ***expression1***, then ***expression2***, and so on.
`add` (concatenation (strings))	**Deprecated** since Flash Player 5. Adobe recommends you use the addition (`+`) operator when creating content for Flash Player 5 or later. **Note**: Flash Lite 2.0 also deprecates the `add` operator in favor of the addition (`+`) operator. Concatenates two or more strings.
`?:` (conditional)	Instructs Flash to evaluate *expression1*, and if the value of *expression1* is `true`, it returns the value of *expression2*; otherwise it returns the value of *expression3*.
`--` (decrement)	A pre-decrement and post-decrement unary operator that subtracts 1 from the *expression*.
`/` (division)	Divides *expression1* by *expression2*.
`/=` (division assignment)	Assigns ***expression1*** the value of ***expression1 / expression2***.
`.` (dot)	Used to navigate movie clip hierarchies to access nested (child) movie clips, variables, or properties.
`==` (equality)	Tests two expressions for equality.
`eq` (equality (strings))	**Deprecated** since Flash Player 5. This operator was deprecated in favor of the `==` `(equality)` operator. Returns `true` if the string representation of *expression1* is equal to the string representation of *expression2*, `false` otherwise.
`>` (greater than)	Compares two expressions and determines whether *expression1* is greater than *expression2*; if it is, the operator returns `true`.

Operator	Description
`gt` (greater than (strings))	**Deprecated** since Flash Player 5. This operator was deprecated in favor of the > (greater than) operator. Compares the string representation of *expression1* with the string representation of *expression2* and returns `true` if *expression1* is greater than *expression2*, `false` otherwise.
`>=` (greater than or equal to)	Compares two expressions and determines whether *expression1* is greater than or equal to *expression2* (`true`) or *expression1* is less than *expression2* (`false`).
`ge` (greater than or equal to (strings))	**Deprecated** since Flash Player 5. This operator was deprecated in favor of the >= (greater than or equal to) operator. Returns `true` if *expression1* is greater than or equal to *expression2*, `false` otherwise.
`++` (increment)	A pre-increment and post-increment unary operator that adds 1 to *expression* .
`!=` (inequality)	Tests for the exact opposite of the equality (`==`) operator.
`<>` (inequality)	**Deprecated** since Flash Player 5. This operator has been deprecated. Adobe recommends that you use the `!= (inequality)` operator. Tests for the exact opposite of the equality (`==`) operator.
instanceof	Tests whether `object` is an instance of `classConstructor` or a subclass of `classConstructor`.
`<` (less than)	Compares two expressions and determines whether *expression1* is less than *expression2*; if so, the operator returns `true`.
`lt` (less than (strings))	**Deprecated** since Flash Player 5. This operator was deprecated in favor of the < (less than) operator. Returns true if expression1 is less than expression2, false otherwise.
`<=` (less than or equal to)	Compares two expressions and determines whether *expression1* is less than or equal to *expression2*; if it is, the operator returns `true`.
`le` (less than or equal to (strings))	**Deprecated** since Flash Player 5. This operator was deprecated in Flash 5 in favor of the <= (less than or equal to) operator. Returns `true` if *expression1* is less than or equal to *expression2*, `false` otherwise.
`//` (line comment delimiter)	Indicates the beginning of a script comment.
`&&` (logical AND)	Performs a Boolean operation on the values of one or both of the expressions.
`and` (logical AND)	**Deprecated** since Flash Player 5. Adobe recommends that you use the logical AND (`&&`) operator. Performs a logical AND (`&&`) operation in Flash Player 4.
`!` (logical NOT)	Inverts the Boolean value of a variable or expression.
`not` (logical NOT)	**Deprecated** since Flash Player 5. This operator was deprecated in favor of the `! (logical NOT)` operator. Performs a logical NOT (!) operation in Flash Player 4.
`||` (logical OR)	Evaluates *expression1* (the expression on the left side of the operator) and returns `true` if the expression evaluates to `true`.
`or` (logical OR)	**Deprecated** since Flash Player 5. This operator was deprecated in favor of the `|| (logical OR)` operator. Evaluates *condition1* and *condition2*, and if either expression is `true`, the whole expression is `true`.

Operator	Description
`%` (modulo)	Calculates the remainder of *expression1* divided by *expression2*.
`%=` (modulo assignment)	Assigns *expression1* the value of *expression1* `%` *expression2*.
`*` (multiplication)	Multiplies two numerical expressions.
`*=` (multiplication assignment)	Assigns *expression1* the value of *expression1* `*` *expression2*.
new	Creates a new, initially anonymous, object and calls the function identified by the `constructor` parameter.
`ne` (not equal (strings))	**Deprecated** since Flash Player 5. This operator was deprecated in favor of the `!=` `(inequality)` operator. Returns `true` if *expression1* is not equal to *expression2*; `false` otherwise.
{} (object initializer)	Creates a new object and initializes it with the specified *name* and *value* property pairs.
() (parentheses)	Performs a grouping operation on one or more parameters, performs sequential evaluation of expressions, or surrounds one or more parameters and passes them as parameters to a function outside the parentheses.
`===` (strict equality)	Tests two expressions for equality; the strict equality (`===`) operator performs in the same way as the equality (`==`) operator, except that data types are not converted.
`!==` (strict inequality)	Tests for the exact opposite of the strict equality (`===`) operator.
`"` (string delimiter)	When used before and after characters, quotation marks (") indicate that the characters have a literal value and are considered a *string*, not a variable, numerical value, or other ActionScript element.
`-` (subtraction)	Used for negating or subtracting.
`-=` (subtraction assignment)	Assigns *expression1* the value of *expression1* `-` *expression2*.
`:` (type)	Used for strict data typing; this operator specifies the variable type, function return type, or function parameter type.
typeof	The `typeof` operator evaluate the `expression` and returns a string specifying whether the expression is a `String, MovieClip, Object, Function, Number,` or `Boolean` value.
void	The `void` operator evaluates an expression and then discards its value, returning `undefined`.

+ addition operator

`expression1 + expression2`

Adds numeric expressions or concatenates (combines) strings. If one expression is a string, all other expressions are converted to strings and concatenated. If both expressions are integers, the sum is an integer; if either or both expressions are floating-point numbers, the sum is a floating-point number.

Note: Flash Lite 2.0 supports the addition (+) operator for adding numeric expressions and concatenating strings. Flash Lite 1.x only supports the addition (+) operator for adding numeric expressions (such as `var1 = 1 + 2 // output: 3`). For Flash Lite 1.x, you must use the `add` operator to concatenate strings.

Availability

Flash Lite 2.0

Operands

`expression1` - A number or string.

`expression2` - A number or string.

Returns

`Object` - A string, integer, or floating-point number.

Example

Usage 1: The following example concatenates two strings and displays the result in the Output panel.

```
var name:String = "Cola";
var instrument:String = "Drums";
trace(name + " plays " + instrument); // output: Cola plays Drums
```

Note: Flash Lite 1.x does not support the addition (+) operator for concatenating strings. For Flash Lite 1.x, you must use the `add` operator to concatenate strings.

Usage 2: This statement adds the integers 2 and 3 and displays the resulting integer, 5, in the Output panel:

```
    trace(2 + 3); // output: 5
```

This statement adds the floating-point numbers 2.5 and 3.25 and displays the resulting floating-point number, 5.75, in the Output panel

```
    trace(2.5 + 3.25); // output: 5.75
```

Usage 3: Variables associated with dynamic and input text fields have the data type String. In the following example, the variable `deposit` is an input text field on the Stage. After a user enters a deposit amount, the script attempts to add `deposit` to `oldBalance`. However, because `deposit` is a String data type, the script concatenates (combines to form one string) the variable values rather than summing them.

```
var oldBalance:Number = 1345.23;
var currentBalance = deposit_txt.text + oldBalance;
trace(currentBalance);
```

For example, if a user enters 475 in the deposit text field, the `trace()` function sends the value 4751345.23 to the Output panel. To correct this, use the `Number()` function to convert the string to a number, as in the following:

```
var oldBalance:Number = 1345.23;
var currentBalance:Number = Number(deposit_txt.text) + oldBalance;
trace(currentBalance);
```

The following example shows how numeric sums to the right of a string expression are not calculated:

```
var a:String = 3 + 10 + "asdf";
trace(a); // 13asdf
var b:String = "asdf" + 3 + 10;
trace(b); // asdf310
```

+= addition assignment operator

`expression1 += expression2`

Assigns *expression1* the value of *expression1*+ *expression2*. For example, the following two statements have the same result:

```
    x += y;
x = x + y;
```

This operator also performs string concatenation. All the rules of the addition (+) operator apply to the addition assignment (+=) operator.

Availability
Flash Lite 1.0

Operands
`expression1 : Number` - A number or string.

`expression2 : Number` - A number or string.

Returns
`Number` - The result of the addition.

Example
Usage 1: This example uses the += operator with a string expression and sends "My name is Gilbert" to the Output panel.

```
var x1:String = "My name is ";
x1 += "Gilbert";
trace(x1); // output: My name is Gilbert
```

Usage 2: The following example shows a numeric use of the addition assignment (+=) operator:

```
var x:Number = 5;
var y:Number = 10;
x += y;
trace(x); // output: 15
```

See also
`+ addition operator`

[] array access operator

```
myArray = [ a0, a1,...aN ]
myArray[ i ] = value
myObject [ propertyName ]
```

Initializes a new array or multidimensional array with the specified elements (`a0`, and so on), or accesses elements in an array. The array access operator lets you dynamically set and retrieve instance, variable, and object names. It also lets you access object properties.

Usage 1: An array is an object whose properties are called *elements*, which are each identified by a number called an *index*. When you create an array, you surround the elements with the array access ([]) operator (or *brackets*). An array can contain elements of various types. For example, the following array, called `employee`, has three elements; the first is a number and the second two are strings (inside quotation marks):

```
var employee:Array = [15, "Barbara", "Jay"];
```

You can nest brackets to simulate multidimensional arrays. You can nest arrays up to 256 levels deep. The following code creates an array called `ticTacToe` with three elements; each element is also an array with three elements:

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]; // Select Debug > List Variables in
test mode
// to see a list of the array elements.
```

Usage 2: Surround the index of each element with brackets ([]) to access it directly; you can add a new element to an array, or you can change or retrieve the value of an existing element. The first index in an array is always 0, as shown in the following example:

```
var my_array:Array = new Array();
my_array[0] = 15;
my_array[1] = "Hello";
my_array[2] = true;
```

You can use brackets ([]) to add a fourth element, as shown in the following example:

```
my_array[3] = "George";
```

You can use brackets ([]) to access an element in a multidimensional array. The first set of brackets identifies the element in the original array, and the second set identifies the element in the nested array. The following lines of code send the number 6 to the Output panel.

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
trace(ticTacToe[1][2]);// output: 6
```

Usage 3: You can use the array access ([]) operator instead of the `eval()` function to dynamically set and retrieve values for movie clip names or any property of an object. The following line of code sets the name of the movie clip determined by concatenating the string "mc" with the value of the `i` variable to "left_corner".

```
name["mc" + i] = "left_corner";
```

Availability

Flash Lite 2.0

Operands

`myArray : Object` - The name of an array.

`a0, a1,...aN : Object` - Elements in an array; any native type or object instance, including nested arrays.

`i : Number` - An integer index greater than or equal to 0.

`myObject : Object` - The name of an object.

`propertyName : String` - A string that names a property of the object.

Returns

`Object` -

Usage 1: A reference to an array.

Usage 2: A value from the array; either a native type or an object instance (including an array instance).

Usage 3: A property from the object; either a native type or an object instance (including an array instance).

Example

The following example shows two ways to create a new empty Array object; the first line uses brackets ([]):

```
var my_array:Array = [];
var my_array:Array = new Array();
```

The following example creates an array called `employee_array` and uses the trace() statement to send the elements to the Output panel. In the fourth line, an element in the array is changed, and the fifth line sends the newly modified array to the Output panel:

```
var employee_array = ["Barbara", "George", "Mary"];
trace(employee_array); // output: Barbara,George,Mary
employee_array[2] = "Sam";
trace(employee_array); // output: Barbara,George,Sam
```

In the following example, the expression inside the brackets (`"piece" + i`) is evaluated and the result is used as the name of the variable to be retrieved from the `my_mc` movie clip. In this example, the variable `i` must live on the same Timeline as the button. If the variable `i` is equal to 5, for example, the value of the variable `piece5` in the `my_mc` movie clip is displayed in the Output panel:

```
myBtn_btn.onRelease = function() {
    x = my_mc["piece"+i];
    trace(x);
};
```

In the following example, the expression inside the brackets is evaluated, and the result is used as the name of the variable to be retrieved from movie clip `name_mc`:

```
name_mc["A" + i];
```

If you are familiar with the Flash 4 ActionScript slash syntax, you can use the `eval()` function to accomplish the same result:

```
eval("name_mc.A" & i);
```

You can use the following ActionScript to loop over all objects in the `_root` scope, which is useful for debugging:

```
for (i in _root) {
    trace(i+": "+_root[i]);
}
```

You can also use the array access ([]) operator on the left side of an assignment statement to dynamically set instance, variable, and object names:

```
employee_array[2] = "Sam";
```

See also

`Array`, `Object`, `eval function`

= assignment operator

`expression1 = expression2`

Assigns the value of *expression2* (the parameter on the right) to the variable, array element, or property in *expression1*. Assignment can be either by value or by reference. Assignment by value copies the actual value of *expression2* and stores it in *expression1*. Assignment by value is used when a variable is assigned a number or string literal. Assignment by reference stores a reference to *expression2* in *expression1*. Assignment by reference is commonly used with the `new` operator. Use of the `new` operator creates an object in memory and a reference to that location in memory is assigned to a variable.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - A variable, element of an array, or property of an object.

`expression2 : Object` - A value of any type.

Returns

`Object` - The assigned value, *expression2*.

Example

The following example uses assignment by value to assign the value of 5 to the variable `x`.

```
var x:Number = 5;
```

The following example uses assignment by value to assign the value "`hello`" to the variable `x`:

```
var x:String;x = " hello ";
```

The following example uses assignment by reference to create the moonsOfJupiter variable, which contains a reference to a newly created Array object. Assignment by value is then used to copy the value "Callisto" to the first element of the array referenced by the variable `moonsOfJupiter`:

```
var moonsOfJupiter:Array = new Array();moonsOfJupiter[0] = "Callisto";
```

The following example uses assignment by reference to create a new object, and assign a reference to that object to the variable `mercury`. Assignment by value is then used to assign the value of `3030` to the `diameter` property of the `mercury` object:

```
var mercury:Object = new Object(); mercury.diameter = 3030; // in miles trace
(mercury.diameter); // output: 3030
```

The following example builds upon the previous example by creating a variable named `merkur` (the German word for mercury) and assigning it the value of `mercury`. This creates two variables that reference the same object in memory, which means you can use either variable to access the object's properties. We can then change the `diameter` property to use kilometers instead of miles:

```
var merkur:Object = mercury; merkur.diameter = 4878; // in kilometers trace (mercury.diameter);
// output: 4878
```

See also

`==` equality operator

& bitwise AND operator

expression1 `&` *expression2*

Converts *expression1* and *expression2* to 32-bit unsigned integers, and performs a Boolean AND operation on each bit of the integer parameters. Floating-point numbers are converted to integers by discarding any digits after the decimal point. The result is a new 32-bit integer.

Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value using the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and then have the most significant digits discarded as well.

The return value is interpreted as a signed two's complement number, so the return is an integer in the range -2147483648 to 2147483647.

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number.

`expression2 : Number` - A number.

Returns

`Number` - The result of the bitwise operation.

Example

The following example compares the bit representation of the numbers and returns 1 only if both bits at the same position are 1. In the following ActionScript code, you add 13 (binary 1101) and 11 (binary 1011) and return 1 only in the position where both numbers have a 1.

```
var insert:Number = 13;
var update:Number = 11;
trace(insert & update); // output : 9 (or 1001 binary)
```

In the numbers 13 and 11 the result is 9 because only the first and last positions in both numbers have the number 1.

The following example shows the behavior of the return value conversion:

```
trace(0xFFFFFFFF); // 4294967295
trace(0xFFFFFFFF & 0xFFFFFFFF); // -1
trace(0xFFFFFFFF & -1); // -1
trace(4294967295 & -1); // -1
trace(4294967295 & 4294967295); // -1
```

See also

`&= bitwise AND assignment operator`, `^ bitwise XOR operator`, `^= bitwise XOR assignment operator`, `| bitwise OR operator`, `|= bitwise OR assignment operator`, `~ bitwise NOT operator`

&= bitwise AND assignment operator

`expression1 &= expression2`

Assigns *expression1* the value of *expression1* `&` *expression2*. For example, the following two expressions are equivalent:

```
x &= y;
x = x & y;
```

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number.

`expression2 : Number` - A number.

Returns

`Number` - The value of *expression1* `&` *expression2*.

Example

The following example assigns the value `9` to `x`:

```
var x:Number = 15;
var y:Number = 9;
trace(x &= y); // output: 9
```

See also

`&` bitwise AND operator, `^` bitwise XOR operator, `^=` bitwise XOR assignment operator, `|` bitwise OR operator `|=` bitwise OR assignment operator, `~` bitwise NOT operator

<< bitwise left shift operator

expression1 `<<` *expression2*

Converts *expression1* and *expression2* to 32-bit integers, and shifts all the bits in *expression1* to the left by the number of places specified by the integer resulting from the conversion of *expression2*. The bit positions that are emptied as a result of this operation are filled in with 0 and bits shifted off the left end are discarded. Shifting a value left by one position is the equivalent of multiplying it by 2.

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number or expression to be shifted left.

`expression2 : Number` - A number or expression that converts to an integer from 0 to 31.

Returns

`Number` - The result of the bitwise operation.

Example

In the following example, the integer 1 is shifted 10 bits to the left: `x = 1 << 10` The result of this operation is `x = 1024`. This is because 1 decimal equals 1 binary, 1 binary shifted left by 10 is 10000000000 binary, and 10000000000 binary is 1024 decimal. In the following example, the integer 7 is shifted 8 bits to the left: `x = 7 << 8` The result of this operation is `x = 1792`. This is because 7 decimal equals 111 binary, 111 binary shifted left by 8 bits is 11100000000 binary, and 11100000000 binary is 1792 decimal. If you trace the following example, you see that the bits have been pushed two spaces to the left:

```
// 2 binary == 0010
// 8 binary == 1000
trace(2 << 2); // output: 8
```

See also

`>>= bitwise right shift and assignment operator`, `>> bitwise right shift operator`, `<<= bitwise left shift and assignment operator>>> bitwise unsigned right shift operator`, `>>>= bitwise unsigned right shift and assignment operator`

<<= bitwise left shift and assignment operator

`expression1 <<= expression2`

This operator performs a bitwise left shift (<<) operation and stores the contents as a result in *expression1*. The following two expressions are equivalent:

```
A <<= BA = (A << B)
```

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number or expression to be shifted left.

`expression2 : Number` - A number or expression that converts to an integer from 0 to 31.

Returns

`Number` - The result of the bitwise operation.

Example

In the following example, you use the bitwise left shift and assignment (<<=) operator to shift all bits one space to the left:

```
    var x:Number = 4;
// shift all bits one slot to the left.
x <<= 1;
trace(x); // output: 8
// 4 decimal = 0100 binary
// 8 decimal = 1000 binary
```

See also

`<< bitwise left shift operator`, `>>= bitwise right shift and assignment operator`, `>> bitwise right shift operator`

~ bitwise NOT operator

`~expression`

Also known as the one's complement operator or the bitwise complement operator. Converts the *expression* to a 32-bit signed integer, and then applies a bitwise one's complement. That is, every bit that is a 0 is set to 1 in the result, and every bit that is a 1 is set to 0 in the result. The result is a signed 32-bit integer.

For example, the hex value 0x7777 is represented as this binary number: 0111011101110111

The bitwise negation of that hex value, ~0x7777, is this binary number: 1000100010001000

In hexadecimal, this is 0x8888. Therefore, ~0x7777 is 0x8888.

The most common use of bitwise operators is for representing *flag bits* (Boolean values packed into 1 bit each).

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value is an integer in the range -2147483648 to 2147483647.

Availability

Flash Lite 2.0

Operands

`expression : Number` - A number.

Returns

`Number` - The result of the bitwise operation.

Example

The following example demonstrates a use of the bitwise NOT (~) operator with flag bits:

```
var ReadOnlyFlag:Number = 0x0001; // defines bit 0 as the read-only flag
var flags:Number = 0;
trace(flags);
/* To set the read-only flag in the flags variable,
    the following code uses the bitwise OR:
*/
flags |= ReadOnlyFlag;
trace(flags);
/* To clear the read-only flag in the flags variable,
    first construct a mask by using bitwise NOT on ReadOnlyFlag.
    In the mask, every bit is a 1 except for the read-only flag.
    Then, use bitwise AND with the mask to clear the read-only flag.
    The following code constructs the mask and performs the bitwise AND:
*/
flags &= ~ReadOnlyFlag;
trace(flags);
// output: 0 1 0
```

See also

`& bitwise AND operator`, `&= bitwise AND assignment operator`, `^ bitwise XOR operator`, `^= bitwise XOR assignment operator`| bitwise OR operator, `|= bitwise OR assignment operator`

| bitwise OR operator

expression1 | *expression2*

Converts *expression1* and *expression2* to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits of either *expression1* or *expression2* are 1. Floating-point numbers are converted to integers by discarding any digits after the decimal point. The result is a new 32-bit integer.

Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

Availability
Flash Lite 2.0

Operands
`expression1 : Number` - A number.

`expression2 : Number` - A number.

Returns
`Number` - The result of the bitwise operation.

Example
The following is an example of a bitwise OR (|) operation:

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 | 1001 = 1111
trace(x | y); // returns 15 decimal (1111 binary)
```

Don't confuse the single | (bitwise OR) with || (logical OR).

See also
`& bitwise AND operator`, `&= bitwise AND assignment operator`, `^ bitwise XOR operator`, `^= bitwise XOR assignment operator`|`= bitwise OR assignment operator`, `~ bitwise NOT operator`

|= bitwise OR assignment operator

`expression1 |= expression2`

Assigns *expression1* the value of *expression1* | *expression2*. For example, the following two statements are equivalent:

```
x |= y; and x = x | y;
```

Availability
Flash Lite 2.0

Operands
`expression1 : Number` - A number or variable.

`expression2 : Number` - A number or variable.

Returns

`Number` - The result of the bitwise operation.

Example

The following example uses the bitwise OR assignment (`|=`) operator:

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 |= 1001 = 1111
trace(x |= y); // returns 15 decimal (1111 binary)
```

See also

`&` bitwise AND operator, `&=` bitwise AND assignment operator, `^` bitwise XOR operator, `^=` bitwise XOR assignment operator| bitwise OR operator, `~` bitwise NOT operator

>> bitwise right shift operator

`expression1 >> expression2`

Converts *expression1* and *expression2* to 32-bit integers, and shifts all the bits in *expression1* to the right by the number of places specified by the integer that results from the conversion of *expression2*. Bits that are shifted off the right end are discarded. To preserve the sign of the original *expression*, the bits on the left are filled in with 0 if the most significant bit (the bit farthest to the left) of *expression1* is 0, and filled in with 1 if the most significant bit is 1. Shifting a value right by one position is the equivalent of dividing by 2 and discarding the remainder.

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number or expression to be shifted right.

`expression2 : Number` - A number or expression that converts to an integer from 0 to 31.

Returns

`Number` - The result of the bitwise operation.

Example

The following example converts 65535 to a 32-bit integer and shifts it 8 bits to the right:

```
var x:Number = 65535 >> 8;
trace(x); // outputs 255
```

The following example shows the result of the previous example:

```
var x:Number = 255;
```

This is because 65535 decimal equals 1111111111111111 binary (sixteen 1s), 1111111111111111 binary shifted right by 8 bits is 11111111 binary, and 11111111 binary is 255 decimal. The most significant bit is 0 because the integers are 32-bit, so the fill bit is 0.

The following example converts -1 to a 32-bit integer and shifts it 1 bit to the right:

```
var x:Number = -1 >> 1;
trace(x); // outputs -1
```

The following example shows the result of the previous example:

```
var x:Number = -1;
```

This is because -1 decimal equals 11111111111111111111111111111111 binary (thirty-two 1s), shifting right by one bit causes the least significant (bit farthest to the right) to be discarded and the most significant bit to be filled in with 1. The result is 11111111111111111111111111111111 (thirty-two 1s) binary, which represents the 32-bit integer -1.

See also

`>>= bitwise right shift and assignment operator`

>>= bitwise right shift and assignment operator

`expression1 >>= expression2`

This operator performs a bitwise right shift operation and stores the contents as a result in *expression1*.

The following two statements are equivalent:

```
A >>= B; and A = (A >> B);
```

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number or expression to be shifted right.

`expression2 : Number` - A number or expression that converts to an integer from 0 to 31.

Returns

`Number` - The result of the bitwise operation.

Example

The following commented code uses the bitwise right shift and assignment (`>>=`) operator.

```
function convertToBinary(numberToConvert:Number):String {
    var result:String = "";
    for (var i = 0; i<32; i++) {
    // Extract least significant bit using bitwise AND
    var lsb:Number = numberToConvert & 1;
    // Add this bit to the result
    string result = (lsb ? "1" : "0")+result;
    // Shift numberToConvert right by one bit, to see next bit
    numberToConvert >>= 1;
    }
    return result;
}
trace(convertToBinary(479));
// Returns the string 00000000000000000000000111011111
// This string is the binary representation of the decimal
// number 479
```

See also

`>>` bitwise right shift operator

>>> bitwise unsigned right shift operator

expression1 `>>>` *expression2*

The same as the bitwise right shift (`>>`) operator except that it does not preserve the sign of the original *expression* because the bits on the left are always filled with 0.

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number or expression to be shifted right.

`expression2 : Number` - A number or expression that converts to an integer between 0 and 31.

Returns

`Number` - The result of the bitwise operation.

Example

The following example converts -1 to a 32-bit integer and shifts it 1 bit to the right:

```
var x:Number = -1 >>> 1;
trace(x); // output: 2147483647
```

This is because -1 decimal is 11111111111111111111111111111111 binary (thirty-two 1s), and when you shift right (unsigned) by 1 bit, the least significant (rightmost) bit is discarded, and the most significant (leftmost) bit is filled with a 0. The result is 01111111111111111111111111111111 binary, which represents the 32-bit integer 2147483647.

See also

`>>= bitwise right shift and assignment operator`

>>>= bitwise unsigned right shift and assignment operator

expression1 `>>>=` *expression2*

Performs an unsigned bitwise right-shift operation and stores the contents as a result in *expression1*. The following two statements are equivalent:

```
A >>>= B; and A = (A >>> B);
```

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number or expression to be shifted right.

`expression2 : Number` - A number or expression that converts to an integer from 0 to 31.

Returns

`Number` - The result of the bitwise operation.

See also

`>>> bitwise unsigned right shift operator`, `>>= bitwise right shift and assignment operator`

^ bitwise XOR operator

expression1 `^` *expression2*

Converts *expression1* and *expression2* to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits in *expression1* or *expression2*, but not both, are 1. Floating-point numbers are converted to integers by discarding any digits after the decimal point. The result is a new 32-bit integer.

Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - A number.

`expression2 : Number` - A number.

Returns

`Number` - The result of the bitwise operation.

Example

The following example uses the bitwise XOR operator on the decimals 15 and 9, and assigns the result to the variable `x`:

```
// 15 decimal = 1111 binary
// 9 decimal = 1001 binary
var x:Number = 15 ^ 9;
trace(x);
// 1111 ^ 1001 = 0110
// returns 6 decimal (0110 binary)
```

See also

`&` bitwise AND operator, `&=` bitwise AND assignment operator, `^=` bitwise XOR assignment operator, `|` bitwise OR operator, `|=` bitwise OR assignment operator, `~` bitwise NOT operator

^= bitwise XOR assignment operator

expression1 `^=` *expression2*

Assigns *expression1* the value of *expression1* `^` *expression2*. For example, the following two statements are equivalent:

```
x ^= y x = x ^ y
```

Availability

Flash Lite 2.0

Operands

`expression1 : Number` - Integers and variables.

`expression2 : Number` - Integers and variables.

Returns

`Number` - The result of the bitwise operation.

Example

The following example shows a bitwise XOR assignment (^=) operation:

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
trace(x ^= y); // returns 6 decimal (0110 binary)
```

See also

`&` bitwise AND operator, `&=` bitwise AND assignment operator, `^` bitwise XOR operator, `|` bitwise OR operator`|=` bitwise OR assignment operator, `~` bitwise NOT operator

/* block comment delimiter operator

```
/* comment */
/* comment
comment */
```

Indicates one or more lines of script comments. Any characters that appear between the opening comment tag (`/*`) and the closing comment tag (`*/`) are interpreted as a comment and ignored by the ActionScript interpreter. Use the // (comment delimiter) to identify single-line comments. Use the /* comment delimiter to identify comments on multiple successive lines. Leaving off the closing tag (`*/`) when using this form of comment delimiter returns an error message. Attempting to nest comments also returns an error message. After an opening comment tag (/*) is used, the first closing comment tag (`*/`) will end the comment, regardless of the number of opening comment tags (/*) placed between them.

Availability

Flash Lite 1.0

Operands

`comment` - Any characters.

Example

The following script uses comment delimiters at the beginning of the script:

```
/* records the X and Y positions of
the ball and bat movie clips */
var ballX:Number = ball_mc._x;
var ballY:Number = ball_mc._y;
var batX:Number = bat_mc._x;
var batY:Number = bat_mc._y;
```

The following attempt to nest comments will result in an error message:

```
/* this is an attempt to nest comments.
/* But the first closing tag will be paired
with the first opening tag */
and this text will not be interpreted as a comment */
```

See also

`// line comment delimiter operator`

, comma operator

`(expression1 , expression2 [, expressionN... ])`

Evaluates *expression1*, then *expression2*, and so on. This operator is primarily used with the `for` loop statement and is often used with the parentheses () operator.

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - An expression to be evaluated.

`expression2 : Number` - An expression to be evaluated.

`expressionN : Number` - Any number of additional expressions to be evaluated.

Returns

`Object` - The value of *expression1*, *expression2*, and so on.

Example

The following example uses the comma (,) operator in a `for` loop:

```
for (i = 0, j = 0; i < 3 && j < 3; i++, j+=2) {
    trace("i = " + i + ", j = " + j);
}
// Output:
// i = 0, j = 0
// i = 1, j = 2
```

The following example uses the comma (,) operator without the parentheses () operator and illustrates that the comma operator returns only the value of the first expression without the parentheses () operator:

```
var v:Number = 0;
v = 4, 5, 6;
trace(v); // output: 4
```

The following example uses the comma (,) operator with the parentheses () operator and illustrates that the comma operator returns the value of the last expression when used with the parentheses () operator:

```
var v:Number = 0;
v = (4, 5, 6);
trace(v); // output: 6
```

The following example uses the comma (,) operator without the parentheses () operator and illustrates that the comma operator sequentially evaluates all of the expressions but returns the value of the first expression. The second expression, `z++`, is evaluated and `z` is incremented by one.

```
var v:Number = 0;
var z:Number = 0;
v = v + 4 , z++, v + 6;
trace(v); // output: 4
trace(z); // output: 1
```

The following example is identical to the previous example except for the addition of the parentheses () operator and illustrates once again that, when used with the parentheses () operator, the comma (,) operator returns the value of the last expression in the series:

```
var v:Number = 0;
var z:Number = 0;
v = (v + 4, z++, v + 6);
trace(v); // output: 6
trace(z); // output: 1
```

See also

`() parentheses operator`

add concatenation (strings) operator

string1 `add` *string2*

Deprecated since Flash Player 5. Adobe recommends you use the addition (+) operator when creating content for Flash Player 5 or later.

Note: Flash Lite 2.0 also deprecates the `add` operator in favor of the addition (+) operator.

Concatenates two or more strings. The add (+) operator replaces the Flash 4 `&` operator; Flash Player 4 files that use the `&` operator are automatically converted to use the add (+) operator for string concatenation when brought into the Flash 5 or later authoring environment. You must use the add (+) operator to concatenate strings if you are creating content for Flash Player 4 or earlier versions of the Flash Player.

Availability
Flash Lite 1.0

Operands
`string1 : String` - A string.

`string2 : String` - A string.

Returns
`String` - The concatenated string.

See also
`+ addition operator`

?: conditional operator

`expression1 ? expression2 : expression3`

Instructs Flash to evaluate *expression1*, and if the value of *expression1* is `true`, it returns the value of *expression2*; otherwise it returns the value of *expression3*.

Availability
Flash Lite 1.0

Operands
`expression1 : Object` - An expression that evaluates to a Boolean value; usually a comparison expression, such as `x < 5`.

`expression2 : Object` - Values of any type.

`expression3 : Object` - Values of any type.

Returns
`Object` - The value of *expression2* or *expression3*.

Example
The following statement assigns the value of variable `x` to variable `z` because `expression1` evaluates to `true`:

```
var x:Number = 5;
var y:Number = 10;
var z = (x < 6) ? x: y;
trace (z); // returns 5
```

The following example shows a conditional statement written in shorthand:

```
var timecode:String = (new Date().getHours() < 11) ? "AM" : "PM";
trace(timecode);
```

The same conditional statement could also be written in longhand, as shown in the following example:

```
    if (new Date().getHours() < 11) {
    var timecode:String = "AM";
} else {
    var timecode:String = "PM";
} trace(timecode);
```

-- decrement operator

```
--expression
expression--
```

A pre-decrement and post-decrement unary operator that subtracts 1 from the *expression*. The *expression* can be a variable, element in an array, or property of an object. The pre-decrement form of the operator (*--expression*) subtracts 1 from *expression* and returns the result. The post-decrement form of the operator (*expression--*) subtracts 1 from the *expression* and returns the initial value of *expression* (the value prior to the subtraction).

Availability
Flash Lite 1.0

Operands
`expression : Number` - A number or a variable that evaluates to a number.

Returns
`Number` - The result of the decremented value.

Example
The pre-decrement form of the operator decrements `x` to 2 (`x - 1 = 2`) and returns the result as `y`:

```
var x:Number = 3;
var y:Number = --x; //y is equal to 2
```

The post-decrement form of the operator decrements `x` to 2 (`x - 1 = 2`) and returns the original value of `x` as the result `y`:

```
var x:Number = 3;
var y:Number = x--; //y is equal to 3
```

The following example loops from 10 to 1, and each iteration of the loop decreases the counter variable `i` by 1.

```
for (var i = 10; i>0; i--) {
    trace(i);
}
```

/ division operator

```
expression1 / expression2
```

Divides *expression1* by *expression2*. The result of the division operation is a double-precision floating-point number.

Availability
Flash Lite 1.0

Operands
`expression : Number` - A number or a variable that evaluates to a number.

Returns
`Number` - The floating-point result of the operation.

Example
The following statement divides the current width and height of the Stage, and then displays the result in the Output panel.

```
trace(Stage.width/2);
trace(Stage.height/2);
```

For a default Stage width and height of 550 x 400, the output is 275 and 150.

See also
`% modulo operator`

/= division assignment operator

`expression1 /= expression2`

Assigns *expression1* the value of *expression1 / expression2*. For example, the following two statements are equivalent:

```
x /= y; and x = x / y;
```

Availability
Flash Lite 1.0

Operands
`expression1 : Number` - A number or a variable that evaluates to a number.

`expression2 : Number` - A number or a variable that evaluates to a number.

Returns
`Number` - A number.

Example
The following code illustrates using the division assignment (`/=`) operator with variables and numbers:

```
var x:Number = 10;
var y:Number = 2;
x /= y; trace(x); // output: 5
```

See also
`/ division operator`

. dot operator

object.property_or_method
instancename.variable
instancename.childinstance
instancename.childinstance.variable

Used to navigate movie clip hierarchies to access nested (child) movie clips, variables, or properties. The dot operator is also used to test or set the properties of an object or top-level class, execute a method of an object or top-level class, or create a data structure.

Availability
Flash Lite 1.0

Operands
`object : Object` - An instance of a class. The object can be an instance of any of the built-in ActionScript classes or a custom class. This parameter is always to the left of the dot (.) operator.

`property_or_method` - The name of a property or method associated with an object. All the valid methods and properties for the built-in classes are listed in the method and property summary tables for that class. This parameter is always to the right of the dot (.) operator.

`instancename : MovieClip` - The instance name of a movie clip.

`variable` — The instance name to the left of the dot (.) operator can also represent a variable on the Timeline of the movie clip.

`childinstance : MovieClip` - A movie clip instance that is a child of, or nested in, another movie clip.

Returns
`Object` - The method, property, or movie clip named on the right side of the dot.

Example
The following example identifies the current value of the variable `hairColor` in the movie clip `person_mc`:

```
person_mc.hairColor
```

The Flash 4 authoring environment did not support dot syntax, but Flash MX 2004 files published for Flash Player 4 can use the dot operator. The preceding example is equivalent to the following (deprecated) Flash 4 syntax:

```
/person_mc:hairColor
```

The following example creates a new movie clip within the `_root` scope. Then a text field is created inside the movie clip called `container_mc`. The text field's `autoSize` property is set to `true` and then populated with the current date.

```
this.createEmptyMovieClip("container_mc", this.getNextHighestDepth());
this.container_mc.createTextField("date_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
this.container_mc.date_txt.autoSize = true;
this.container_mc.date_txt.text = new Date();
```

The dot (.) operator is used when targeting instances within the SWF file and when you need to set properties and values for those instances.

== equality operator

expression1 == expression2

Tests two expressions for equality. The result is `true` if the expressions are equal.

The definition of equal depends on the data type of the parameter:

- Numbers and Boolean values are compared by value and are considered equal if they have the same value.
- String expressions are equal if they have the same number of characters and the characters are identical.
- Variables representing objects, arrays, and functions are compared by reference. Two such variables are equal if they refer to the same object, array, or function. Two separate arrays are never considered equal, even if they have the same number of elements.

When comparing by value, if *expression1* and *expression2* are different data types, ActionScript will attempt to convert the data type of *expression2* to match that of *expression1*.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - A number, string, Boolean value, variable, object, array, or function.

`expression2 : Object` - A number, string, Boolean value, variable, object, array, or function.

Returns

`Boolean` - The Boolean result of the comparison.

Example

The following example uses the equality (`==`) operator with an `if` statement:

```
    var a:String = "David", b:String = "David";
if (a == b) {
    trace("David is David");
}
```

The following examples show the results of operations that compare mixed types:

```
    var x:Number = 5;
var y:String = "5";
trace(x == y); // output: true
var x:String = "5";
var y:String = "66";
trace(x == y); // output: false
var x:String = "chris";
var y:String = "steve";
trace(x == y); // output: false
```

The following examples show comparison by reference. The first example compares two arrays with identical length and elements. The equality operator will return false for these two arrays. Although the arrays appear equal, comparison by reference requires that they both refer to the same array. The second example creates the thirdArray variable, which points to the same array as the variable firstArray. The equality operator will return true for these two arrays because the two variables refer to the same array.

```
    var firstArray:Array = new Array("one", "two", "three");
var secondArray:Array = new Array("one", "two", "three");
trace(firstArray == secondArray);
// will output false
// Arrays are only considered equal
// if the variables refer to the same array.
var thirdArray:Array = firstArray;
trace(firstArray == thirdArray); // will output true
```

See also

`! logical NOT operator`, `!= inequality operator`, `!== strict inequality operator`, `&& logical AND operator`, `|| logical OR operator`, `=== strict equality operator`

eq equality (strings) operator

expression1 `eq` *expression2*

Deprecated since Flash Player 5. This operator was deprecated in favor of the `== (equality)` operator.

Compares two expressions for equality and returns a value of `true` if the string representation of *expression1* is equal to the string representation of *expression2*, `false` otherwise.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - Numbers, strings, or variables.

`expression2 : Object` - Numbers, strings, or variables.

Returns

`Boolean` - The result of the comparison.

See also

`== equality operator`

> greater than operator

expression1 `>` *expression2*

Compares two expressions and determines whether *expression1* is greater than *expression2*; if it is, the operator returns `true`. If *expression1* is less than or equal to *expression2*, the operator returns `false`. String expressions are evaluated using alphabetical order; all capital letters come before lowercase letters.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - A number or string.

`expression2 : Object` - A number or string.

Returns

`Boolean` - The Boolean result of the comparison.

Example

In the following example, the greater than (>) operator is used to determine whether the value of the text field `score_txt` is greater than 90:

```
    if (score_txt.text>90) {
    trace("Congratulations, you win!");
} else {
    trace("sorry, try again");
}
```

gt greater than (strings) operator

expression1 `gt` *expression2*

Deprecated since Flash Player 5. This operator was deprecated in favor of the > (greater than) operator.

Compares the string representation of *expression1* with the string representation of *expression2* and returns `true` if *expression1* is greater than *expression2*, `false` otherwise.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - Numbers, strings, or variables.

`expression2 : Object` - Numbers, strings, or variables.

Returns

`Boolean` - The Boolean result of the comparison.

See also

`> greater than operator`

>= greater than or equal to operator

expression1 `>=` *expression2*

Compares two expressions and determines whether *expression1* is greater than or equal to *expression2* (`true`) or *expression1* is less than *expression2* (`false`).

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - A string, integer, or floating-point number.

`expression2 : Object` - A string, integer, or floating-point number.

Returns

`Boolean` - The Boolean result of the comparison.

Example

In the following example, the greater than or equal to (>=) operator is used to determine whether the current hour is greater than or equal to 12:

```
if (new Date().getHours() >= 12) {
    trace("good afternoon");
} else {
    trace("good morning");
}
```

ge greater than or equal to (strings) operator

expression1 `ge` *expression2*

Deprecated since Flash Player 5. This operator was deprecated in favor of the >= (greater than or equal to) operator.

Compares the string representation of *expression1* with the string representation of *expression2* and returns `true` if *expression1* is greater than or equal to *expression2*, `false` otherwise.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - Numbers, strings, or variables.

`expression2 : Object` - Numbers, strings, or variables.

Returns

`Boolean` - The result of the comparison.

See also

`>= greater than or equal to operator`

++ increment operator

++expression
`expression++`

A pre-increment and post-increment unary operator that adds 1 to *expression*. The *expression* can be a variable, element in an array, or property of an object. The pre-increment form of the operator (*++expression*) adds 1 to *expression* and returns the result. The post-increment form of the operator (*expression*++) adds 1 to *expression* and returns the initial value of *expression* (the value prior to the addition).

The pre-increment form of the operator increments `x` to 2 (`x + 1 = 2`) and returns the result as `y`:

```
var x:Number = 1;
var y:Number = ++x;
trace("x:"+x); //traces x:2
trace("y:"+y); //traces y:2
```

The post-increment form of the operator increments `x` to 2 (`x + 1 = 2`) and returns the original value of `x` as the result `y`:

```
var x:Number = 1;
var y:Number = x++;
trace("x:"+x); //traces x:2
trace("y:"+y); //traces y:1
```

Availability

Flash Lite 1.0

Operands

`expression : Number` - A number or a variable that evaluates to a number.

Returns

`Number` - The result of the increment.

Example

The following example uses ++ as a post-increment operator to make a `while` loop run five times:

```
var i:Number = 0;
while (i++ < 5) {
    trace("this is execution " + i);
}
/* output:
    this is execution 1
    this is execution 2
    this is execution 3
    this is execution 4
    this is execution 5
*/
```

The following example uses ++ as a pre-increment operator:

```
var a:Array = new Array();
var i:Number = 0;
while (i < 10) {
    a.push(++i);
}
trace(a.toString()); //traces: 1,2,3,4,5,6,7,8,9,10
```

This example also uses ++ as a pre-increment operator.

```
var a:Array = [];
for (var i = 1; i <= 10; ++i) {
    a.push(i);
}
trace(a.toString()); //traces: 1,2,3,4,5,6,7,8,9,10
```

This script shows the following result in the Output panel: `1,2,3,4,5,6,7,8,9,10`

The following example uses ++ as a post-increment operator in a `while` loop:

```
// using a while loop
var a:Array = new Array();
var i:Number = 0;
while (i < 10) {
    a.push(i++);
}
trace(a.toString()); //traces 0,1,2,3,4,5,6,7,8,9
```

The following example uses ++ as a post-increment operator in a `for` loop:

```
// using a for loop
var a:Array = new Array();
for (var i = 0; i < 10; i++) {
    a.push(i);
}
trace(a.toString()); //traces 0,1,2,3,4,5,6,7,8,9
```

This script displays the following result in the Output panel:

```
0,1,2,3,4,5,6,7,8,9
```

!= inequality operator

expression1 `!=` *expression2*

Tests for the exact opposite of the equality (`==`) operator. If *expression1* is equal to *expression2* , the result is `false`. As with the equality (`==`) operator, the definition of equal depends on the data types being compared, as illustrated in the following list:

- Numbers, strings, and Boolean values are compared by value.
- Objects, arrays, and functions are compared by reference.
- A variable is compared by value or by reference, depending on its type.

Comparison by value means what most people would expect equals to mean –that two expressions have the same value. For example, the expression (2 + 3) is equal to the expression (1 + 4) when compared by value.

Comparison by reference means that two expressions are equal only if they both refer to the same object, array, or function. Values inside the object, array, or function are not compared.

When comparing by value, if *expression1* and *expression2* are different data types, ActionScript will attempt to convert the data type of *expression2* to match that of *expression1*.

Availability

Flash Lite 2.0

Operands

`expression1 : Object` - A number, string, Boolean value, variable, object, array, or function.

`expression2 : Object` - A number, string, Boolean value, variable, object, array, or function.

Returns

`Boolean` - The Boolean result of the comparison.

Example

The following example illustrates the result of the inequality (`!=`) operator:

```
trace(5 != 8); // returns true
trace(5 != 5); //returns false
```

The following example illustrates the use of the inequality (`!=`) operator in an `if` statement:

```
var a:String = "David";
var b:String = "Fool";
if (a != b) {
    trace("David is not a fool");
}
```

The following example illustrates comparison by reference with two functions:

```
var a:Function = function() { trace("foo"); };
var b:Function = function() { trace("foo"); };
a(); // foo
b(); // foo
trace(a != b); // true
a = b;
a(); // foo
b(); // foo
trace(a != b); // false
// trace statement output: foo foo true foo foo false
```

The following example illustrates comparison by reference with two arrays:

```
var a:Array = [ 1, 2, 3 ];
var b:Array = [ 1, 2, 3 ];
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a!=b); // true
a = b;
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a != b); // false
// trace statement output: 1,2,3 1,2,3 true 1,2,3 1,2,3 false
```

See also

`!` logical NOT operator, `!==` strict inequality operator, `&&` logical AND operator, `||` logical OR operator`==` equality operator, `===` strict equality operator

<> inequality operator

expression1 `<>` *expression2*

Deprecated since Flash Player 5. This operator has been deprecated. Adobe recommends that you use the `!=` (inequality) operator.

Tests for the exact opposite of the equality (`==`) operator. If *expression1* is equal to *expression2*, the result is `false`. As with the equality (`==`) operator, the definition of equal depends on the data types being compared:

- Numbers, strings, and Boolean values are compared by value.
- Objects, arrays, and functions are compared by reference.
- Variables are compared by value or by reference depending on their type.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - A number, string, Boolean value, variable, object, array, or function.

`expression2 : Object` - A number, string, Boolean value, variable, object, array, or function.

Returns

`Boolean` - The Boolean result of the comparison.

See also

`!= inequality operator`

instanceof operator

`object` `instanceof` *`classConstructor`*

Tests whether `object` is an instance of `classConstructor` or a subclass of `classConstructor`. The `instanceof` operator does not convert primitive types to wrapper objects. For example, the following code returns `true`:

```
new String("Hello") instanceof String;
```

Whereas the following code returns false:

```
"Hello" instanceof String;
```

Availability

Flash Lite 2.0

Operands

`object : Object` - An ActionScript object.

`classConstructor : Function` - A reference to an ActionScript constructor function, such as `String` or `Date`.

Returns

`Boolean` - If `object` is an instance of or a subclass of `classConstructor`, `instanceof` returns `true`, otherwise it returns `false`. Also, `_global instanceof Object` returns `false`.

See also

`typeof operator`

< less than operator

`expression1` `<` *`expression2`*

Compares two expressions and determines whether *expression1* is less than *expression2*; if so, the operator returns `true`. If *expression1* is greater than or equal to *expression2*, the operator returns `false`. String expressions are evaluated using alphabetical order; all capital letters come before lowercase letters.

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - A number or string.

`expression2 : Number` - A number or string.

Returns

`Boolean` - The Boolean result of the comparison.

Example

The following examples show `true` and `false` returns for both numeric and string comparisons:

```
trace(3 < 10); // true
trace(10 < 3); // false
trace("Allen" < "Jack"); // true
trace("Jack" < "Allen"); //false
trace("11" < "3"); // true
trace("11" < 3); // false (numeric comparison)
trace("C" < "abc"); // true
trace("A" < "a"); // true
```

lt less than (strings) operator

expression1 `lt` *expression2*

Deprecated since Flash Player 5. This operator was deprecated in favor of the < (less than) operator.

Compares *expression1* to *expression2* and returns `true` if *expression1* is less than *expression2*, `false` otherwise.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - Numbers, strings, or variables.

`expression2 : Object` - Numbers, strings, or variables.

Returns

`Boolean` - The result of the comparison.

See also

`< less than operator`

<= less than or equal to operator

expression1 `<=` *expression2*

Compares two expressions and determines whether *expression1* is less than or equal to *expression2*; if it is, the operator returns `true`. If *expression1* is greater than *expression2*, the operator returns `false`. String expressions are evaluated using alphabetical order; all capital letters come before lowercase letters.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - A number or string.

`expression2 : Object` - A number or string.

Returns

`Boolean` - The Boolean result of the comparison.

Example

The following examples show `true` and `false` results for both numeric and string comparisons:

```
    trace(5 <= 10); // true
trace(2 <= 2); // true
trace(10 <= 3); // false
trace("Allen" <= "Jack"); // true
trace("Jack" <= "Allen"); // false
trace("11" <= "3"); // true
trace("11" <= 3); // false (numeric comparison)
trace("C" <= "abc"); // true
trace("A" <= a); // true
```

le less than or equal to (strings) operator

expression1 `le` *expression2*

Deprecated since Flash Player 5. This operator was deprecated in Flash 5 in favor of the <= (less than or equal to) operator.

Compares *expression1* to *expression2* and returns a value of `true` if *expression1* is less than or equal to *expression2*, `false` otherwise.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - Numbers, strings, or variables.

`expression2 : Object` - Numbers, strings, or variables.

Returns

`Boolean` - The result of the comparison.

See also

`<= less than or equal to operator`

// line comment delimiter operator

`// comment`

Indicates the beginning of a script comment. Any characters that appear between the comment delimiter (`//`) and the end-of-line character are interpreted as a comment and ignored by the ActionScript interpreter.

Availability

Flash Lite 1.0

Operands

`comment` - Any characters.

Example

The following script uses comment delimiters to identify the first, third, fifth, and seventh lines as comments:

```
// record the X position of the ball movie clip
var ballX:Number = ball_mc._x;
// record the Y position of the ball movie clip
var ballY:Number = ball_mc._y;
// record the X position of the bat movie clip
var batX:Number = bat_mc._x;
// record the Y position of the ball movie clip
var batY:Number = bat_mc._y;
```

See also

`/* block comment delimiter operator`

&& logical AND operator

`expression1 && expression2`

Performs a Boolean operation on the values of one or both of the expressions. Evaluates *expression1* (the expression on the left side of the operator) and returns `false` if the expression evaluates to `false`. If *expression1* evaluates to `true`, *expression2* (the expression on the right side of the operator) is evaluated. If *expression2* evaluates to `true`, the final result is `true`; otherwise, it is `false`. The expression `true&&true` evaluates to true, `true&&false` evaluates to false, `false&&false` evaluates to false, and `false&&true` evaluates to false

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - A Boolean value or an expression that converts to a Boolean value.

`expression2 : Number` - A Boolean value or an expression that converts to a Boolean value.

Returns

`Boolean` - A Boolean result of the logical operation.

Example

The following example uses the logical AND (`&&`) operator to perform a test to determine if a player has won the game. The `turns` variable and the `score` variable are updated when a player takes a turn or scores points during the game. The script shows "You Win the Game!" in the Output panel when the player's score reaches 75 or higher in 3 turns or less.

```
var turns:Number = 2;
var score:Number = 77;
if ((turns <= 3) && (score >= 75)) {
    trace("You Win the Game!");
} else {
    trace("Try Again!");
}
// output: You Win the Game!
```

See also

`! logical NOT operator`, `!= inequality operator`, `!== strict inequality operator`, `|| logical OR operator`, `== equality operator`, `=== strict equality operator`

and logical AND operator

condition1 and *condition2*

Deprecated since Flash Player 5. Adobe recommends that you use the logical AND (`&&`) operator.

Performs a logical AND (`&&`) operation in Flash Player 4. If both expressions evaluate to `true`, the entire expression is `true`.

Availability

Flash Lite 1.0

Operands

`condition1 : Boolean` - A condition or expression that evaluates to `true` or `false`.

`condition2 : Boolean` - A condition or expression that evaluates to `true` or `false`.

Returns

`Boolean` - A Boolean result of the logical operation.

See also

`&& logical AND operator`

! logical NOT operator

`! expression`

Inverts the Boolean value of a variable or expression. If *expression* is a variable with the absolute or converted value `true`, the value of `!`*expression* is `false`. If the expression `x && y` evaluates to `false`, the expression `!(x && y)` evaluates to `true`. Therefore, `!true` returns `false`, and `!false` returns `true`.

Availability

Flash Lite 1.0

Operands

`expression : Boolean` - An expression or a variable that evaluates to a Boolean value.

Returns

`Boolean` - The Boolean result of the logical operation.

Example

In the following example, the variable `happy` is set to `false`. The `if` condition evaluates the condition `!happy`, and if the condition is `true`, the trace() statement sends a string to the Output panel.

```
var happy:Boolean = false;
if (!happy) {
    trace("don't worry, be happy"); //traces don't worry, be happy
}
```

The statement traces because `!false` equals `true`.

See also

`!= inequality operator`, `!== strict inequality operator`, `&& logical AND operator`, `|| logical OR operator`, `== equality operator`, `=== strict equality operator`

not logical NOT operator

`not` *`expression`*

Deprecated since Flash Player 5. This operator was deprecated in favor of the `!` (logical NOT) operator.

Performs a logical NOT (`!`) operation in Flash Player 4.

Availability

Flash Lite 1.0

Operands

`expression : Object` - A variable or other expression that converts to a Boolean value.

Returns

`Boolean` - The result of the logical operation.

See also

`! logical NOT operator`

|| logical OR operator

`expression1` `||` *`expression2`*

Evaluates *expression1* (the expression on the left side of the operator) and returns `true` if the expression evaluates to `true`. If *expression1* evaluates to `false`, *expression2* (the expression on the right side of the operator) is evaluated. If *expression2* evaluates to `false`, the final result is `false`; otherwise, it is `true`.

If you use a function call as *expression2*, the function will not be executed by that call if *expression1* evaluates to true.

The result is `true` if either or both expressions evaluate to `true`; the result is `false` only if both expressions evaluate to `false`. You can use the logical OR operator with any number of operands; if any operand evaluates to `true`, the result is `true`.

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - A Boolean value or an expression that converts to a Boolean value.

`expression2 : Number` - A Boolean value or an expression that converts to a Boolean value.

Returns

`Boolean` - The result of the logical operation.

Example

The following example uses the logical OR (||) operator in an `if` statement. The second expression evaluates to `true`, so the final result is `true`:

```
var x:Number = 10;
var y:Number = 250;
var start:Boolean = false;
if ((x > 25) || (y > 200) || (start)) {
    trace("the logical OR test passed"); // output: the logical OR test passed
}
```

The message the logical OR test passed appears because one of the conditions in the `if` statement is true (`y>200`). Although the other two expressions evaluate to `false`, the `if` block is executed because one condition evaluates to `true`.

The following example demonstrates how using a function call as *expression2* can lead to unexpected results. If the expression on the left of the operator evaluates to `true`, that result is returned without evaluating the expression on the right (the function `fx2()` is not called).

```
function fx1():Boolean {
    trace("fx1 called");
    return true;
}
function fx2():Boolean {
    trace("fx2 called");
    return true;
}
if (fx1() || fx2()) {
    trace("IF statement entered");
}
/* The following is sent to the Output panel: /* The following is sent to the log file: fx1
called IF statement entered */
```

See also

`! logical NOT operator`, `!= inequality operator`, `!== strict inequality operator`, `&& logical AND operator`, `== equality operator`, `=== strict equality operator`

or logical OR operator

condition1 `or` *condition2*

Deprecated since Flash Player 5. This operator was deprecated in favor of the `|| (logical OR)` operator.

Evaluates *condition1* and *condition2*, and if either expression is `true`, the whole expression is `true`.

Availability

Flash Lite 1.0

Operands

`condition1 : Boolean` - An expression that evaluates to `true` or `false`.

`condition2 : Boolean` - An expression that evaluates to `true` or `false`.

Returns

`Boolean` - The result of the logical operation.

See also

`||` logical OR operator, `|` bitwise OR operator

% modulo operator

expression1 `%` *expression2*

Calculates the remainder of *expression1* divided by *expression2*. If either of the *expression* parameters are non-numeric, the modulo (`%`) operator attempts to convert them to numbers. The *expression* can be a number or string that converts to a numeric value.

The sign of the result of modulo operation matches the sign of the dividend (the first number). For example, `-4 % 3` and `-4 % -3` both evaluate to `-1`.

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - A number or expression that evaluates to a number.

`expression2 : Number` - A number or expression that evaluates to a number.

Returns

`Number` - The result of the arithmetic operation.

Example

The following numeric example uses the modulo (`%`) operator:

```
trace(12%5); // traces 2
trace(4.3%2.1); // traces 0.0999999999999996
trace(4%4); // traces 0
```

The first trace returns 2, rather than 12/5 or 2.4, because the modulo (`%`) operator returns only the remainder. The second trace returns 0.0999999999999996 instead of the expected 0.1 because of the limitations of floating-point accuracy in binary computing.

See also

`/` division operator, `round (Math.round method)`

%= modulo assignment operator

expression1 `%=` *expression2*

Assigns *expression1* the value of *expression1* `%` *expression2*. The following two statements are equivalent:

```
x %= y; and x = x % y;
```

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - A number or expression that evaluates to a number.

`expression2 : Number` - A number or expression that evaluates to a number.

Returns

`Number` - The result of the arithmetic operation.

Example

The following example assigns the value `4` to the variable `x`:

```
var x:Number = 14;
var y:Number = 5;
trace(x %= y); // output: 4
```

See also

`% modulo operator`

* multiplication operator

`expression1 * expression2`

Multiplies two numerical expressions. If both expressions are integers, the product is an integer. If either or both expressions are floating-point numbers, the product is a floating-point number.

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - A number or expression that evaluates to a number.

`expression2 : Number` - A number or expression that evaluates to a number.

Returns

`Number` - An integer or floating-point number.

Example

Usage 1: The following statement multiplies the integers 2 and 3:

```
trace(2*3); // output: 6
```

The result, 6, is an integer. Usage 2: This statement multiplies the floating-point numbers 2.0 and 3.1416:

```
trace(2.0 * 3.1416); // output: 6.2832
```

The result, 6.2832, is a floating-point number.

*= multiplication assignment operator

`expression1 *= expression2`

Assigns *expression1* the value of *`expression1 * expression2`*. For example, the following two expressions are equivalent:

```
x *= y x = x * y
```

Availability

Flash Lite 1.0

Operands

`expression1 : Number` - A number or expression that evaluates to a number.

`expression2 : Number` - A number or expression that evaluates to a number.

Returns

`Number` - The value of *expression1* `*` *expression2*. If an expression cannot be converted to a numeric value, it returns `NaN` (not a number).

Example

Usage 1: The following example assigns the value 50 to the variable `x`:

```
var x:Number = 5;
var y:Number = 10;
trace(x *= y); // output: 50
```

Usage 2: The second and third lines of the following example calculate the expressions on the right side of the equal sign and assign the results to `x` and `y`:

```
var i:Number = 5;
var x:Number = 4 - 6;
var y:Number = i + 2;
trace(x *= y); // output: -14
```

See also

`*` multiplication operator

new operator

`new constructor()`

Creates a new, initially anonymous, object and calls the function identified by the `constructor` parameter. The `new` operator passes to the function any optional parameters in parentheses, as well as the newly created object, which is referenced using the keyword `this`. The `constructor` function can then use this to set the variables of the object.

Availability

Flash Lite 2.0

Operands

`constructor : Object` - A function followed by any optional parameters in parentheses. The function is usually the name of the object type (for example, `Array`, `Number`, or `Object`) to be constructed.

Example

The following example creates the `Book()` function and then uses the `new` operator to create the objects `book1` and `book2`.

```
function Book(name, price){
    this.name = name;
    this.price = price;
}

book1 = new Book("Confederacy of Dunces", 19.95);
book2 = new Book("The Floating Opera", 10.95);
```

The following example uses the `new` operator to create an `Array` object with 18 elements:

```
golfCourse_array = new Array(18);
```

See also

`[] array access operator`, `{} object initializer operator`

ne not equal (strings) operator

`expression1 ne expression2`

Deprecated since Flash Player 5. This operator was deprecated in favor of the `!= (inequality)` operator.

Compares *expression1* to *expression2* and returns `true` if *expression1* is not equal to *expression2*; `false` otherwise.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - Numbers, strings, or variables.

`expression2 : Object` - Numbers, strings, or variables.

Returns

`Boolean` - Returns `true` if `expression1` is not equal to `expression2`; `false` otherwise.

See also

`!= inequality operator`

{} object initializer operator

```
object = { name1 : value1 , name2 : value2 ,... nameN : valueN }
{expression1; [...expressionN]}
```

Creates a new object and initializes it with the specified *name* and *value* property pairs. Using this operator is the same as using the `new Object` syntax and populating the property pairs using the assignment operator. The prototype of the newly created object is generically named the Object object.

This operator is also used to mark blocks of contiguous code associated with flow control statements (`for`, `while`, `if`, `else`, `switch`) and functions.

Availability

Flash Lite 2.0

Operands

`object : Object` - The object to create. *name1,2,...N* The names of the properties. *value1,2,...N* The corresponding values for each *name* property.

Returns

`Object` -

Usage 1: An Object object.

Usage 2: Nothing, except when a function has an explicit `return` statement, in which case the return type is specified in the function implementation.

Example

The first line of the following code creates an empty object using the object initializer ({}) operator; the second line creates a new object using a constructor function:

```
var object:Object = {};
var object:Object = new Object();
```

The following example creates an object `account` and initializes the properties `name`, `address`, `city`, `state`, `zip`, and `balance` with accompanying values:

```
var account:Object = {name:"Macromedia, Inc.", address:"600 Townsend Street", city:"San
Francisco", state:"California", zip:"94103", balance:"1000"};
for (i in account) {
    trace("account." + i + " = " + account[i]);
}
```

The following example shows how array and object initializers can be nested within each other:

```
var person:Object = {name:"Gina Vechio", children:["Ruby", "Chickie", "Puppa"]};
```

The following example uses the information in the previous example and produces the same result using constructor functions:

```
var person:Object = new Object();
person.name = "Gina Vechio";
person.children = new Array();
person.children[0] = "Ruby";
person.children[1] = "Chickie";
person.children[2] = "Puppa";
```

The previous ActionScript example can also be written in the following format:

```
var person:Object = new Object();
person.name = "Gina Vechio";
person.children = new Array("Ruby", "Chickie", "Puppa");
```

See also

`Object`

() parentheses operator

```
(expression1 [, expression2])
( expression1, expression2 )
function ( parameter1,..., parameterN )
```

Performs a grouping operation on one or more parameters, performs sequential evaluation of expressions, or surrounds one or more parameters and passes them as parameters to a function outside the parentheses.

Usage 1: Controls the order in which the operators execute in the expression. Parentheses override the normal precedence order and cause the expressions within the parentheses to be evaluated first. When parentheses are nested, the contents of the innermost parentheses are evaluated before the contents of the outer ones.

Usage 2: Evaluates a series of expressions, separated by commas, in sequence, and returns the result of the final expression.

Usage 3: Surrounds one or more parameters and passes them as parameters to the function outside the parentheses.

Availability

Flash Lite 1.0

Operands

`expression1 : Object` - Numbers, strings, variables, or text.

`expression2 : Object` - Numbers, strings, variables, or text.

`function : Function` - The function to be performed on the contents of the parentheses.

`parameter1...parameterN : Object` - A series of parameters to execute before the results are passed as parameters to the function outside the parentheses.

Example

Usage 1: The following statements show the use of parentheses to control the order in which expressions are executed (the value of each expression appears in the Output panel):

```
trace((2 + 3)*(4 + 5)); // displays 45
trace((2 + 3) * (4 + 5)); // writes 45
trace(2 + (3 * (4 + 5))); // displays 29
trace(2 + (3 * (4 + 5))); // writes 29
trace(2+(3*4)+5); // displays 19
trace(2 + (3 * 4) + 5); // writes19
```

Usage 2: The following example evaluates the function `foo()`, and then the function `bar()`, and returns the result of the expression `a + b`:

```
var a:Number = 1;
var b:Number = 2;
function foo() { a += b; }
function bar() { b *= 10; }
trace((foo(), bar(), a + b)); // outputs 23
```

Usage 3: The following example shows the use of parentheses with functions:

```
var today:Date = new Date();
trace(today.getFullYear()); // traces current year
function traceParameter(param):Void { trace(param); }
traceParameter(2 * 2); //traces 4
```

See also

`with statement`

=== strict equality operator

`expression1 === expression2`

Tests two expressions for equality; the strict equality (`===`) operator performs in the same way as the equality (`==`) operator, except that data types are not converted. The result is `true` if both expressions, including their data types, are equal.

The definition of equal depends on the data type of the parameter:

- Numbers and Boolean values are compared by value and are considered equal if they have the same value.
- String expressions are equal if they have the same number of characters and the characters are identical.

- Variables representing objects, arrays, and functions are compared by reference. Two such variables are equal if they refer to the same object, array, or function. Two separate arrays are never considered equal, even if they have the same number of elements.

Availability

Flash Lite 2.0

Operands

`expression1 : Object` - A number, string, Boolean value, variable, object, array, or function.

`expression2 : Object` - A number, string, Boolean value, variable, object, array, or function.

Returns

`Boolean` - The Boolean result of the comparison.

Example

The comments in the following code show the returned value of operations that use the equality and strict equality operators:

```
    // Both return true because no conversion is done
var string1:String = "5";
var string2:String = "5";
trace(string1 == string2); // true
trace(string1 === string2); // true
// Automatic data typing in this example converts 5 to "5"
var string1:String = "5";
var num:Number = 5;
trace(string1 == num); // true
trace(string1 === num); // false
// Automatic data typing in this example converts true to "1"
var string1:String = "1";
var bool1:Boolean = true;
trace(string1 == bool1); // true
trace(string1 === bool1); // false
// Automatic data typing in this example converts false to "0"
var string1:String = "0";
var bool2:Boolean = false;
trace(string1 == bool2); // true
trace(string1 === bool2); // false
```

The following examples show how strict equality treats variables that are references differently than it treats variables that contain literal values. This is one reason to consistently use String literals and to avoid the use of the `new` operator with the String class.

```
    // Create a string variable using a literal value
var str:String = "asdf";
// Create a variable that is a reference
var stringRef:String = new String("asdf");
// The equality operator does not distinguish among literals, variables,
// and references
trace(stringRef == "asdf"); // true
trace(stringRef == str); // true
trace("asdf" == str); // true
// The strict equality operator considers variables that are references
// distinct from literals and variables
trace(stringRef === "asdf"); // false
trace(stringRef === str); // false
```

See also

`! logical NOT operator`, `!= inequality operator`, `!== strict inequality operator`, `&& logical AND operator`, `|| logical OR operator`, `== equality operator`

!== strict inequality operator

expression1 `!==` *expression2*

Tests for the exact opposite of the strict equality (`===`) operator. The strict inequality operator performs the same as the inequality operator except that data types are not converted.

If *expression1* is equal to *expression2*, and their data types are equal, the result is `false`. As with the strict equality (`===`) operator, the definition of equal depends on the data types being compared, as illustrated in the following list:

- Numbers, strings, and Boolean values are compared by value.
- Objects, arrays, and functions are compared by reference.
- A variable is compared by value or by reference, depending on its type.

Availability

Flash Lite 2.0

Operands

`expression1 : Object` - A number, string, Boolean value, variable, object, array, or function.

`expression2 : Object` - A number, string, Boolean value, variable, object, array, or function.

Returns

`Boolean` - The Boolean result of the comparison.

Example

The comments in the following code show the returned value of operations that use the equality (`==`), strict equality (`===`), and strict inequality (`!==`) operators:

```
var s1:String = "5";
var s2:String = "5";
var s3:String = "Hello";
var n:Number = 5;
var b:Boolean = true;
trace(s1 == s2); // true
trace(s1 == s3); // false
trace(s1 == n); // true
trace(s1 == b); // false
trace(s1 === s2); // true
trace(s1 === s3); // false
trace(s1 === n); // false
trace(s1 === b); // false
trace(s1 !== s2); // false
trace(s1 !== s3); // true
trace(s1 !== n); // true
trace(s1 !== b); // true
```

See also

`!` logical NOT operator, `!=` inequality operator, `&&` logical AND operator, `||` logical OR operator, `==` equality operator, `===` strict equality operator

" string delimiter operator

"text"

When used before and after characters, quotation marks (") indicate that the characters have a literal value and are considered a *string*, not a variable, numerical value, or other ActionScript element.

Availability

Flash Lite 1.0

Operands

`text : String` - A sequence of zero or more characters.

Example

The following example uses quotation marks (") to indicate that the value of the variable *yourGuess* is the literal string `"Prince Edward Island"` and not the name of a variable. The value of `province` is a variable, not a literal; to determine the value of `province`, the value of *yourGuess* must be located.

```
    var yourGuess:String = "Prince Edward Island";
submit_btn.onRelease = function() { trace(yourGuess); };
// displays Prince Edward Island in the Output panel
// writes Prince Edward Island to the log file
```

See also

`String`, `String function`

- subtraction operator

(Negation) -expression
(Subtraction) expression1 - expression2

Used for negating or subtracting.

Usage 1: When used for negating, it reverses the sign of the numerical `expression`.

Usage 2: When used for subtracting, it performs an arithmetic subtraction on two numerical expressions, subtracting *expression2* from *expression1*. When both expressions are integers, the difference is an integer. When either or both expressions are floating-point numbers, the difference is a floating-point number.

Availability
Flash Lite 1.0

Operands
`expression1 : Number` - A number or expression that evaluates to a number.

`expression2 : Number` - A number or expression that evaluates to a number.

Returns
`Number` - An integer or floating-point number.

Example
Usage 1: The following statement reverses the sign of the expression 2 + 3:

```
trace(-(2+3)); // output: -5
```

Usage 2: The following statement subtracts the integer 2 from the integer 5:

```
trace(5-2); // output: 3
```

The result, 3, is an integer.

The following statement subtracts the floating-point number 1.5 from the floating-point number 3.25:

```
trace(3.25-1.5); // output: 1.75
```

The result, 1.75, is a floating-point number.

-= subtraction assignment operator

`expression1 -= expression2`

Assigns *expression1* the value of *expression1* - *expression2*. For example, the following two statements are equivalent:

```
x -= y ; x = x - y;
```

String expressions must be converted to numbers; otherwise, `NaN` (not a number) is returned.

Availability
Flash Lite 1.0

Operands
`expression1 : Number` - A number or expression that evaluates to a number.

`expression2 : Number` - A number or expression that evaluates to a number.

Returns
`Number` - The result of the arithmetic operation.

Example

The following example uses the subtraction assignment (`-=`) operator to subtract 10 from 5 and assign the result to the variable `x`:

```
    var x:Number = 5;
var y:Number = 10;
x -= y; trace(x); // output: -5
```

The following example shows how strings are converted to numbers:

```
    var x:String = "5";
var y:String = "10";
x -= y; trace(x); // output: -5
```

See also

`- subtraction operator`

: type operator

```
[ modifiers ] var variableName : type
function functionName () : type { ... }
function functionName ( parameter1:type , ... , parameterN:type ) [ :type ]{ ... }
```

Used for strict data typing; this operator specifies the variable type, function return type, or function parameter type. When used in a variable declaration or assignment, this operator specifies the variable's type; when used in a function declaration or definition, this operator specifies the function's return type; when used with a function parameter in a function definition, this operator specifies the variable type expected for that parameter.

Types are a compile-time-only feature. All types are checked at compile time, and errors are generated when there is a mismatch. Mismatches can occur during assignment operations, function calls, and class member dereferencing using the dot (`.`) operator. To avoid type mismatch errors, use strict data typing.

Types that you can use include all native object types, classes and interfaces that you define, and Function and Void. The recognized native types are Boolean, Number, and String. All built-in classes are also supported as native types.

Availability

Flash Lite 2.0

Operands

`variableName : Object` - An identifier for a variable.

`type` : A native data type, class name that you have defined, or interface name.

`functionName` : An identifier for a function.

`parameter` : An identifier for a function parameter.

Example

Usage 1: The following example declares a public variable named `userName` whose type is String and assigns an empty string to it:

```
var userName:String = "";
```

Usage 2: The following example shows how to specify a function's parameter type by defining a function named `randomInt()` that takes a parameter named `integer` of type Number:

```
function randomInt(integer:Number):Number {
    return Math.round(Math.random()*integer);
}
trace(randomInt(8));
```

Usage 3: The following example defines a function named `squareRoot()` that takes a parameter named `val` of the Number type and returns the square root of `val`, also a Number type:

```
function squareRoot(val:Number):Number {
    return Math.sqrt(val);
}
trace(squareRoot(121));
```

See also

`set variable statement`, `Array function`

typeof operator

`typeof(`*`expression`*`)`

The `typeof` operator evaluates the `expression` and returns a string specifying whether the expression is a `String`, `MovieClip`, `Object`, `Function`, `Number`, or `Boolean` value.

Availability

Flash Lite 2.0

Operands

`expression : Object` - A string, movie clip, button, object, or function.

Returns

`String` - A `String` representation of the type of `expression`. The following table shows the results of the `typeof` operator on each type of `expression`.

Expression Type	Result
String	`string`
Movie clip	`movieclip`
Button	`object`
Text field	`object`
Number	`number`
Boolean	`boolean`
Object	`object`
Function	`function`

See also

`instanceof operator`

void operator

`void` *`expression`*

The `void` operator evaluates an expression and then discards its value, returning `undefined`. The `void` operator is often used in comparisons using the `==` operator to test for undefined values.

Availability

Flash Lite 2.0

Operands

`expression : Object` - An expression to be evaluated.

Statements

Statements are language elements that perform or specify an action. For example, the `return` statement returns a result as a value of the function in which it executes. The `if` statement evaluates a condition to determine the next action that should be taken. The `switch` statement creates a branching structure for ActionScript statements.

Statements summary

Statement	Description
break	Appears within a loop (`for`, `for..in`, `do..while`, or `while`) or within a block of statements associated with a particular case within a `switch` statement.
case	Defines a condition for the `switch` statement.
class	Defines a custom class, which lets you instantiate objects that share methods and properties that you define.
continue	Jumps past all remaining statements in the innermost loop and starts the next iteration of the loop as if control had passed through to the end of the loop normally.
default	Defines the default case for a `switch` statement.
delete	Destroys the object reference specified by the ***`reference`*** parameter, and returns `true` if the reference is successfully deleted; `false` otherwise.
do..while	Similar to a `while` loop, except that the statements are executed once before the initial evaluation of the condition.
dynamic	Specifies that objects based on the specified class can add and access dynamic properties at runtime.
else	Specifies the statements to run if the condition in the `if` statement returns `false`.
else if	Evaluates a condition and specifies the statements to run if the condition in the initial `if` statement returns `false`.
extends	Defines a class that is a subclass of another class; the latter is the superclass.
for	Evaluates the ***`init`*** (initialize) expression once and then starts a looping sequence.
for..in	Iterates over the properties of an object or elements in an array and executes the ***`statement`*** for each property or element.
function	Comprises a set of statements that you define to perform a certain task.

Statement	Description
get	Permits implicit *getting* of properties associated with objects based on classes you have defined in external class files.
if	Evaluates a condition to determine the next action in a SWF file.
implements	Specifies that a class must define all the methods declared in the interface (or interfaces) being implemented.
import	Lets you access classes without specifying their fully qualified names.
interface	Defines an interface.
intrinsic	Allows compile-time type checking of previously defined classes.
private	Specifies that a variable or function is available only to the class that declares or defines it or to subclasses of that class.
public	Specifies that a variable or function is available to any caller.
return	Specifies the value returned by a function.
set	Permits implicit setting of properties associated with objects based on classes you have defined in external class files.
set variable	Assigns a value to a variable.
static	Specifies that a variable or function is created only once per class rather than being created in every object based on that class.
super	Invokes the superclass version of a method or constructor.
switch	Creates a branching structure for ActionScript statements.
throw	Generates, or *throws*, an error that can be handled, or *caught*, by a `catch{}` code block.
try..catch..finally	Enclose a block of code in which an error can occur, and then respond to the error.
var	Used to declare local or Timeline variables.
while	Evaluates a condition and if the condition evaluates to `true`, runs a statement or series of statements before looping back to evaluate the condition again.
with	Lets you specify an object (such as a movie clip) with the ***`object`*** parameter and evaluate expressions and actions inside that object with the ***`statement(s)`*** parameter.

break statement

`break`

Appears within a loop (`for`, `for..in`, `do..while`, or `while`) or within a block of statements associated with a particular case within a `switch` statement. When used in a loop, the `break` statement instructs Flash to skip the rest of the loop body, stop the looping action, and execute the statement following the loop statement. When used in a `switch`, the `break` statement instructs Flash to skip the rest of the statements in that `case` block and jump to the first statement following the enclosing `switch` statement.

In nested loops, the `break` statement only skips the rest of the immediate loop and does not break out of the entire series of nested loops. For breaking out of an entire series of nested loops, see `try..catch..finally`.

Availability

Flash Lite 1.0

Example

The following example uses the `break` statement to exit an otherwise infinite loop:

```
var i:Number = 0;
while (true) {
    trace(i);
    if (i >= 10) {
    break; // this will terminate/exit the loop
    }
    i++;
}
```

which traces the following output:

```
0
1
2
3
4
5
6
7
8
9
10
```

See also

`_forceframerate property`

case statement

`case` *`expression`* `:` *`statement(s)`*

Defines a condition for the `switch` statement. If the *expression* parameter equals the *expression* parameter of the `switch` statement using strict equality (`===`), then Flash Player will execute statements in the *statement(s)* parameter until it encounters a `break` statement or the end of the `switch` statement.

If you use the `case` statement outside a `switch` statement, it produces an error and the script doesn't compile.

Note: You should always end the *statement(s)* parameter with a `break` statement. If you omit the break statement from the *statement(s)* parameter, it continues executing with the next `case` statement instead of exiting the `switch` statement.

Availability

Flash Lite 1.0

Parameters

expression : `String` - Any expression.

Example

The following example defines conditions for the `switch` statement `thisMonth`. If `thisMonth` equals the expression in the `case` statement, the statement executes.

```
var thisMonth:Number = new Date().getMonth();
switch (thisMonth) {
    case 0 :
    trace("January");
    break;
    case 1 :
    trace("February");
    break;
    case 5 :
    case 6 :
    case 7 :
    trace("Some summer month");
    break;
    case 8 :
    trace("September");
    break;
    default :
    trace("some other month");
}
```

See also

`break statement`

class statement

`[dynamic] class` *`className`* `[ extends` *`superClass`* `] [ implements` *`interfaceName`*`[, `*`interfaceName`*`... ] ] { // class definition here}`

Defines a custom class, which lets you instantiate objects that share methods and properties that you define. For example, if you are developing an invoice-tracking system, you could create an invoice class that defines all the methods and properties that each invoice should have. You would then use the `new invoice()` command to create invoice objects.

The name of the class must match the name of the external file that contains the class. The name of the external file must be the name of the class with the file extension .as appended. For example, if you name a class Student, the file that defines the class must be named Student.as.

If a class is within a package, the class declaration must use the fully qualified class name of the form base.sub1.sub2.MyClass. Also, the class's AS file must be stored within the path in a directory structure that reflects the package structure, such as base/sub1/sub2/MyClass.as. If a class definition is of the form "class MyClass," it is in the default package and the MyClass.as file should be in the top level of some directory in the path.

For this reason, it's good practice to plan your directory structure before you begin creating classes. Otherwise, if you decide to move class files after you create them, you have to modify the class declaration statements to reflect their new location.

You cannot nest class definitions; that is, you cannot define additional classes within a class definition.

To indicate that objects can add and access dynamic properties at runtime, precede the class statement with the `dynamic` keyword. To declare that a class implements an interface, use the `implements` keyword. To create subclasses of a class, use the `extends` keyword. (A class can extend only one class, but can implement several interfaces.) You can use `implements` and `extends` in a single statement. The following examples show typical uses of the `implements` and `extends` keywords:

```
class C implements Interface_i, Interface_j // OK
class C extends Class_d implements Interface_i, Interface_j // OK
class C extends Class_d, Class_e // not OK
```

Availability

Flash Lite 2.0

Parameters

className : `String` - The fully qualified name of the class.

Example

The following example creates a class called Plant. The Plant constructor takes two parameters.

```
// Filename Plant.as
class Plant {
    // Define property names and types
    var leafType:String;
    var bloomSeason:String;
    // Following line is constructor
    // because it has the same name as the class
    function Plant(param_leafType:String, param_bloomSeason:String) {
    // Assign passed values to properties when new Plant object is created
    this.leafType = param_leafType;
    this.bloomSeason = param_bloomSeason;
    }
    // Create methods to return property values, because best practice
    // recommends against directly referencing a property of a class
    function getLeafType():String {
    return leafType;
    }
    function getBloomSeason():String {
    return bloomSeason;
    }
}
```

In an external script file or in the Actions panel, use the `new` operator to create a Plant object.

```
var pineTree:Plant = new Plant("Evergreen", "N/A");
// Confirm parameters were passed correctly
trace(pineTree.getLeafType());
trace(pineTree.getBloomSeason());
```

The following example creates a class called ImageLoader. The `ImageLoader` constructor takes three parameters.

```
// Filename ImageLoader.as
class ImageLoader extends MovieClip {
    function ImageLoader(image:String, target_mc:MovieClip, init:Object) {
    var listenerObject:Object = new Object();
    listenerObject.onLoadInit = function(target) {
    for (var i in init) {
    target[i] = init[i];
    }
    };
    var JPEG_mcl:MovieClipLoader = new MovieClipLoader();
    JPEG_mcl.addListener(listenerObject);
    JPEG_mcl.loadClip(image, target_mc);
    }
}
```

In an external script file or in the Actions panel, use the `new` operator to create an `ImageLoader` object.

```
var jakob_mc:MovieClip = this.createEmptyMovieClip("jakob_mc", this.getNextHighestDepth());
var jakob:ImageLoader = new
ImageLoader("http://www.helpexamples.com/flash/images/image1.jpg", jakob_mc, {_x:10, _y:10,
_alpha:70, _rotation:-5});
```

See also

`dynamic` statement

continue statement

```
continue
```

Jumps past all remaining statements in the innermost loop and starts the next iteration of the loop as if control had passed through to the end of the loop normally. It has no effect outside a loop.

Availability

Flash Lite 1.0

Example

In the following `while` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump to the top of the loop, where the condition is tested:

```
trace("example 1");
var i:Number = 0;
while (i < 10) {
    if (i % 3 == 0) {
    i++;
    continue;
    }
    trace(i);
    i++;
}
```

In the following `do..while` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump to the bottom of the loop, where the condition is tested:

```
trace("example 2");
var i:Number = 0;
do {
    if (i % 3 == 0) {
    i++;
    continue;
    }
    trace(i);
    i++;
}
while (i < 10);
```

In a `for` loop, `continue` causes the Flash interpreter to skip the rest of the loop body. In the following example, if the i modulo 3 equals 0, then the `trace(i)` statement is skipped:

```
trace("example 3");
for (var i = 0; i < 10; i++) {
    if (i % 3 == 0) {
    continue;
    }
    trace(i);
}
```

In the following `for..in` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump back to the top of the loop, where the next value in the enumeration is processed:

```
for (i in _root) {
    if (i == "$version") {
    continue;
    }
    trace(i);
}
```

See also

`do..while statement`

default statement

`default: statements`

Defines the default case for a `switch` statement. The statements execute if the *expression* parameter of the `switch` statement doesn't equal (using the strict equality [`===`] operation) any of the *expression* parameters that follow the `case` keywords for a given `switch` statement.

A `switch` is not required to have a `default` case statement. A `default` case statement does not have to be last in the list. If you use a `default` statement outside a `switch` statement, it produces an error and the script doesn't compile.

Availability

Flash Lite 2.0

Parameters

statements: `String` - Any statements.

Example

In the following example, the expression A does not equal the expressions B or D, so the statement following the `default` keyword is run and the `trace()` statement is sent to the Output panel.

```
var dayOfWeek:Number = new Date().getDay();
switch (dayOfWeek) {
    case 1 :
    trace("Monday");
    break;
    case 2 :
    trace("Tuesday");
    break;
    case 3 :
    trace("Wednesday");
    break;
    case 4 :
    trace("Thursday");
    break;
    case 5 :
    trace("Friday");
    break;
    default :
    trace("Weekend");

}
```

See also

`switch statement`

delete statement

`delete` *reference*

Destroys the object reference specified by the *reference* parameter, and returns `true` if the reference is successfully deleted; `false` otherwise. This operator is useful for freeing memory used by scripts. You can use the `delete` operator to remove references to objects. After all references to an object are removed, Flash Player takes care of removing the object and freeing the memory used by that object.

Although `delete` is an operator, it is typically used as a statement, as shown in the following example:

```
delete x;
```

The `delete` operator can fail and return `false` if the *reference* parameter does not exist or cannot be deleted. You cannot delete predefined objects and properties, nor can you delete variables that are declared within a function with the `var` statement. You cannot use the `delete` operator to remove movie clips.

Availability

Flash Lite 2.0

Returns

`Boolean` - A Boolean value.

Parameters

reference : `Object` - The name of the variable or object to eliminate.

Example

Usage 1: The following example creates an object, uses it, and deletes it after it is no longer needed:

```
var account:Object = new Object();
account.name = "Jon";
account.balance = 10000;
trace(account.name); //output: Jon
delete account;
trace(account.name); //output: undefined
```

Usage 2: The following example deletes a property of an object:

```
// create the new object "account"
var account:Object = new Object();
// assign property name to the account
account.name = "Jon";
// delete the property
delete account.name;
```

Usage 3: The following example deletes an object property:

```
var my_array:Array = new Array();
my_array[0] = "abc"; // my_array.length == 1
my_array[1] = "def"; // my_array.length == 2
my_array[2] = "ghi"; // my_array.length == 3
// my_array[2] is deleted, but Array.length is not changed
delete my_array[2];
trace(my_array.length); // output: 3
trace(my_array); // output: abc,def,undefined
```

Usage 4: The following example shows the behavior of `delete` on object references:

```
var ref1:Object = new Object();
ref1.name = "Jody";
// copy the reference variable into a new variable
// and delete ref1
ref2 = ref1;
delete ref1;
trace("ref1.name "+ref1.name); //output: ref1.name undefined
trace("ref2.name "+ref2.name); //output: ref2.name Jody
```

If `ref1` had not been copied into `ref2`, the object would have been deleted when `ref1` was deleted because there would be no references to it. If you delete `ref2`, there are no references to the object; it will be destroyed, and the memory it used becomes available.

See also

`set variable statement`

do..while statement

`do { statement(s) } while (condition)`

Similar to a `while` loop, except that the statements are executed once before the initial evaluation of the condition. Subsequently, the statements are executed only if the condition evaluates to `true`.

A `do..while` loop ensures that the code inside the loop executes at least once. Although this can also be done with a `while` loop by placing a copy of the statements to be executed before the `while` loop begins, many programmers believe that `do..while` loops are easier to read.

If the condition always evaluates to `true`, the `do..while` loop is infinite. If you enter an infinite loop, you encounter problems with Flash Player and eventually get a warning message or crash the player. Whenever possible, you should use a `for` loop if you know the number of times you want to loop. Although `for` loops are easy to read and debug, they cannot replace `do..while` loops in all circumstances.

Availability

Flash Lite 1.0

Parameters

condition : `Boolean` - The condition to evaluate. The *statement(s)* within the `do` block of code will execute as long as the *condition* parameter evaluates to `true`.

Example

The following example uses a `do..while` loop to evaluate whether a condition is `true`, and traces `myVar` until `myVar` is greater than 5. When `myVar` is greater than 5, the loop ends.

```
var myVar:Number = 0;
do {
    trace(myVar);
    myVar++;
}
while (myVar < 5);
/* output:
0
1
2
3
4
*/
```

See also

`break statement`

dynamic statement

```
dynamic class className [ extends superClass ] [ implements interfaceName[, interfaceName...
] ] { // class definition here }
```

Specifies that objects based on the specified class can add and access dynamic properties at runtime.

Type checking on dynamic classes is less strict than type checking on nondynamic classes, because members accessed inside the class definition and on class instances are not compared with those defined in the class scope. Class member functions, however, can still be type checked for return type and parameter types. This behavior is especially useful when you work with MovieClip objects, because there are many different ways of adding properties and objects to a movie clip dynamically, such as `MovieClip.createEmptyMovieClip()` and `MovieClip.createTextField()`.

Subclasses of dynamic classes are also dynamic.

Availability

Flash Lite 2.0

Example

In the following example, class `Person2` has not yet been marked as dynamic, so calling an undeclared function on it generates an error at compile time:

```
class Person2 {
    var name:String;
    var age:Number;
    function Person2(param_name:String, param_age:Number) {
    trace ("anything");
    this.name = param_name;
    this.age = param_age;
    }
}
```

In a FLA or AS file that's in the same directory, add the following ActionScript to Frame 1 on the Timeline:

```
// Before dynamic is added
var craig:Person2 = new Person2("Craiggers", 32);
for (i in craig) {
    trace("craig." + i + " = " + craig[i]);
}
/* output:
craig.age = 32
craig.name = Craiggers */
```

If you add an undeclared function, `dance`, an error is generated, as shown in the following example:

```
trace("");
craig.dance = true;
for (i in craig) {
    trace("craig." + i + " = " + craig[i]);
}
/* output: **Error** Scene=Scene 1, layer=Layer 1, frame=1:Line 14: There is no property with
the name 'dance'. craig.dance = true; Total ActionScript Errors: 1 Reported Errors: 1 */
```

Add the `dynamic` keyword to the Person2 class, so that the first line appears as follows:

```
dynamic class Person2 {
```

Test the code again, and you see the following output:

```
craig.dance = true craig.age = 32 craig.name = Craiggers
```

See also

`class statement`

else statement

`if (condition){ statement(s); } else { statement(s); }`

Specifies the statements to run if the condition in the `if` statement returns `false`. The curly braces ({}) used to enclose the block of statements to be executed by the `else` statement are not necessary if only one statement will execute.

Availability

Flash Lite 1.0

Parameters

condition : `Boolean` - An expression that evaluates to `true` or `false`.

Example

In the following example, the `else` condition is used to check whether the `age_txt` variable is greater than or less than 18:

```
if (age_txt.text>=18) { trace("welcome, user"); } else { trace("sorry, junior");
userObject.minor = true; userObject.accessAllowed = false; }
```

In the following example, curly braces ({}) are not necessary because only one statement follows the `else` statement:

```
if (age_txt.text>18) { trace("welcome, user"); } else trace("sorry, junior");
```

See also

`ifFrameLoaded function`

else if statement

```
if (condition){ statement(s); }
    else if (condition){ statement(s);}
```

Evaluates a condition and specifies the statements to run if the condition in the initial `if` statement returns `false`. If the `else if` condition returns `true`, the Flash interpreter runs the statements that follow the condition inside curly braces ({}). If the `else if` condition is `false`, Flash skips the statements inside the curly braces and runs the statements following the curly braces.

Use the `else if` statement to create branching logic in your scripts. If there are multiple branches, you should consider using a `switch` statement.

Availability

Flash Lite 1.0

Parameters

condition : `Boolean` - An expression that evaluates to `true` or `false`.

Example

The following example uses `else if` statements to compare `score_txt` to a specified value:

```
if (score_txt.text>90) { trace("A"); } else if (score_txt.text>75) { trace("B"); } else if
(score_txt.text>60) { trace("C"); } else { trace("F"); }
```

See also

`ifFrameLoaded function`

extends statement

Usage 1:

```
class className extends otherClassName {}
```

Usage 2:

```
interface interfaceName extends otherInterfaceName {}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Defines a class that is a subclass of another class; the latter is the superclass. The subclass inherits all the methods, properties, functions, and so on that are defined in the superclass.

Interfaces can also be extended using the `extends` keyword. An interface that extends another interface includes all the original interface's method declarations.

Availability
Flash Lite 2.0

Parameters
className : `String` - The name of the class you are defining.

Example
In the following example, the Car class extends the Vehicle class so that all its methods, properties, and functions are inherited. If your script instantiates a Car object, methods from both the Car class and the Vehicle class can be used.

The following example shows the contents of a file called Vehicle.as, which defines the Vehicle class:

```
class Vehicle {
    var numDoors:Number;
    var color:String;
    function Vehicle(param_numDoors:Number, param_color:String) {
    this.numDoors = param_numDoors;
    this.color = param_color;
    }
    function start():Void {
    trace("[Vehicle] start");
    }
    function stop():Void {
    trace("[Vehicle] stop");
    }
    function reverse():Void {
    trace("[Vehicle] reverse");
    }
}
```

The following example shows a second AS file, called Car.as, in the same directory. This class extends the Vehicle class, modifying it in three ways. First, the Car class adds a variable `fullSizeSpare` to track whether the car object has a full-size spare tire. Second, it adds a new method specific to cars, `activateCarAlarm()`, that activates the car's anti-theft alarm. Third, it overrides the `stop()` function to add the fact that the Car class uses an anti-lock braking system to stop.

```
class Car extends Vehicle {
    var fullSizeSpare:Boolean;
    function Car(param_numDoors:Number, param_color:String, param_fullSizeSpare:Boolean) {
    this.numDoors = param_numDoors;
    this.color = param_color;
    this.fullSizeSpare = param_fullSizeSpare;
    }
    function activateCarAlarm():Void {
    trace("[Car] activateCarAlarm");
    }
    function stop():Void {
    trace("[Car] stop with anti-lock brakes");
    }
}
```

The following example instantiates a Car object, calls a method defined in the Vehicle class (`start()`), then calls the method overridden by the Car class (`stop()`), and finally calls a method from the Car class (`activateCarAlarm()`):

```
var myNewCar:Car = new Car(2, "Red", true);
myNewCar.start(); // output: [Vehicle] start
myNewCar.stop(); // output: [Car] stop with anti-lock brakes
myNewCar.activateCarAlarm(); // output: [Car] activateCarAlarm
```

A subclass of the Vehicle class can also be written using the keyword `super`, which the subclass can use to access properties and methods of the superclass. The following example shows a third AS file, called Truck.as, again in the same directory. The Truck class uses the `super` keyword in the constructor and again in the overridden `reverse()` function.

```
class Truck extends Vehicle {
    var numWheels:Number;
    function Truck(param_numDoors:Number, param_color:String, param_numWheels:Number) {
    super(param_numDoors, param_color);
    this.numWheels = param_numWheels;
    }
    function reverse():Void {
    beep();
    super.reverse();
    }
    function beep():Void {
    trace("[Truck] make beeping sound");
    }
}
```

The following example instantiates a Truck object, calls a method overridden by the Truck class (`reverse()`), then calls a method defined in the Vehicle class (`stop()`):

```
var myTruck:Truck = new Truck(2, "White", 18);
myTruck.reverse(); // output: [Truck] make beeping sound [Vehicle] reverse
myTruck.stop(); // output: [Vehicle] stop
```

See also

`class statement`

for statement

```
    for(init; condition; next) {
statement(s);
}
```

Evaluates the `init` (initialize) expression once and then starts a looping sequence. The looping sequence begins by evaluating the `condition` expression. If the `condition` expression evaluates to true, `statement` is executed and the `next` expression is evaluated. The looping sequence then begins again with the evaluation of the `condition` expression.

The curly braces ({}) used to enclose the block of statements to be executed by the `for` statement are not necessary if only one statement will execute.

Availability

Flash Lite 1.0

Parameters

init - An expression to evaluate before beginning the looping sequence; usually an assignment expression. A `var` statement is also permitted for this parameter.

Example

The following example uses `for` to add the elements in an array:

```
var my_array:Array = new Array();
for (var i:Number = 0; i < 10; i++) {
    my_array[i] = (i + 5) * 10;
}
trace(my_array); // output: 50,60,70,80,90,100,110,120,130,140
```

The following example uses `for` to perform the same action repeatedly. In the code, the `for` loop adds the numbers from 1 to 100.

```
var sum:Number = 0;
for (var i:Number = 1; i <= 100; i++) {
    sum += i;
}
trace(sum); // output: 5050
```

The following example shows that curly braces ({}) are not necessary if only one statement will execute:

```
var sum:Number = 0;
for (var i:Number = 1; i <= 100; i++)
    sum += i;
trace(sum); // output: 5050
```

See also

`++ increment operator`

for..in statement

```
    for (variableIterant in object) { ]
statement(s);
}
```

Iterates over the properties of an object or elements in an array and executes the `statement` for each property or element. Methods of an object are not enumerated by the `for..in` action.

Some properties cannot be enumerated by the `for..in` action. For example, movie clip properties, such as `_x` and `_y`, are not enumerated. In external class files, static members are not enumerable, unlike instance members.

The `for..in` statement iterates over properties of objects in the iterated object's prototype chain. Properties of the object are enumerated first, then properties of its immediate prototype, then properties of the prototype's prototype, and so on. The `for..in` statement does not enumerate the same property name twice. If the object `child` has prototype `parent` and both contain the property `prop`, the `for..in` statement called on `child` enumerates `prop` from `child` but ignores the one in `parent`.

The curly braces ({}) used to enclose the block of statements to be executed by the `for..in` statement are not necessary if only one statement will execute.

If you write a `for..in` loop in a class file (an external AS file), then instance members are not available for the loop, but static members are. However, if you write a `for..in` loop in a FLA file for an instance of the class, then instance members are available but static ones are not.

Availability
Flash Lite 2.0

Parameters
variableIterant : `String` - The name of a variable to act as the iterant, referencing each property of an object or element in an array.

Example
The following example shows using `for..in` to iterate over the properties of an object:

```
var myObject:Object = {firstName:"Tara", age:27, city:"San Francisco"};
for (var prop in myObject) {
    trace("myObject."+prop+" = "+myObject[prop]);
}
//output
myObject.firstName = Tara
myObject.age = 27
myObject.city = San Francisco
```

The following example shows using `for..in` to iterate over the elements of an array:

```
var myArray:Array = new Array("one", "two", "three");
for (var index in myArray)
    trace("myArray["+index+"] = " + myArray[index]);
// output:
myArray[2] = three
myArray[1] = two
myArray[0] = one
```

The following example uses the `typeof` operator with `for..in` to iterate over a particular type of child:

```
for (var name in this) {
    if (typeof (this[name]) == "movieclip") {
    trace("I have a movie clip child named "+name);
    }
}
```

Note: If you have several movie clips, the output consists of the instance names of those clips.

The following example enumerates the children of a movie clip and sends each to Frame 2 in their respective Timelines. The `RadioButtonGroup` movie clip is a parent with several children, `_RedRadioButton_`, `_GreenRadioButton_`, and `_BlueRadioButton_`.

```
for (var name in RadioButtonGroup) { RadioButtonGroup[name].gotoAndStop(2); }
```

function statement

Usage 1: (Declares a named function.)

```
function functionname([parameter0, parameter1,...parameterN]){statement(s)}
```

Usage 2: (Declares an anonymous function and returns a reference to it.)

```
function ([parameter0, parameter1,...parameterN]){ statement(s) }
```

Comprises a set of statements that you define to perform a certain task. You can define a function in one location and invoke, or *call*, it from different scripts in a SWF file. When you define a function, you can also specify parameters for the function. Parameters are placeholders for values on which the function operates. You can pass different parameters to a function each time you call it so you can reuse a function in different situations.

Use the `return` statement in a function's *statement(s)* to cause a function to generate, or *return*, a value.

You can use this statement to define a `function` with the specified *functionname*, *parameters*, and *statement(s)*. When a script calls a function, the statements in the function's definition are executed. Forward referencing is permitted; within the same script, a function may be declared after it is called. A function definition replaces any prior definition of the same function. You can use this syntax wherever a statement is permitted.

You can also use this statement to create an anonymous function and return a reference to it. This syntax is used in expressions and is particularly useful for installing methods in objects.

For additional functionality, you can use the `arguments` object in your function definition. Some common uses of the `arguments` object are creating a function that accepts a variable number of parameters and creating a recursive anonymous function.

Availability

Flash Lite 2.0

Returns

`String` - Usage 1: The declaration form does not return anything. Usage 2: A reference to the anonymous function.

Parameters

functionname : `String` - The name of the declared function.

Example

The following example defines the function `sqr`, which accepts one parameter and returns the `Math.pow(x, 2)` of the parameter:

```
function sqr(x:Number) {
    return Math.pow(x, 2);
}
var y:Number = sqr(3);
trace(y); // output: 9
```

If the function is defined and used in the same script, the function definition may appear after using the function:

```
var y:Number = sqr(3);
trace(y); // output: 9
function sqr(x:Number) {
    return Math.pow(x, 2);
}
```

The following function creates a LoadVars object and loads params.txt into the SWF file. When the file successfully loads, `variables loaded` traces:

```
var myLV:LoadVars = new LoadVars();
myLV.load("params.txt");
myLV.onLoad = function(success:Boolean) {
    trace("variables loaded");
}
```

get statement

```
function get property () { // your statements here }
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Permits implicit *getting* of properties associated with objects based on classes you have defined in external class files. Using implicit get methods lets you access properties of objects without accessing the property directly. Implicit get/set methods are syntactic shorthand for the `Object.addProperty()` method in ActionScript 1.

Availability

Flash Lite 2.0

Parameters

property: `String` - The word you use to refer to the property that `get` accesses; this value must be the same as the value used in the corresponding `set` command.

Example

In the following example, you define a Team class. The Team class includes get/set methods that let you retrieve and set properties within the class:

```
class Team {
    var teamName:String;
    var teamCode:String;
    var teamPlayers:Array = new Array();
    function Team(param_name:String, param_code:String) {
    this.teamName = param_name;
    this.teamCode = param_code;
    }
    function get name():String {
    return this.teamName;
    }
    function set name(param_name:String):Void {
    this.teamName = param_name;
    }
}
```

Enter the following ActionScript in a frame on the Timeline:

```
var giants:Team = new Team("San Fran", "SFO");
trace(giants.name);
giants.name = "San Francisco";
trace(giants.name);
/* output:
San Fran San Francisco */
```

When you trace `giants.name`, you use the `get` method to return the value of the property.

See also

`addProperty (Object.addProperty method)`

if statement

`if(condition) { statement(s); }`

Evaluates a condition to determine the next action in a SWF file. If the condition is `true`, Flash runs the statements that follow the condition inside curly braces ({}). If the condition is `false`, Flash skips the statements inside the curly braces and runs the statements following the curly braces. Use the `if` statement along with the `else` and `else if` statements to create branching logic in your scripts.

The curly braces ({}) used to enclose the block of statements to be executed by the `if` statement are not necessary if only one statement will execute.

Availability

Flash Lite 1.0

Parameters

condition : `Boolean` - An expression that evaluates to `true` or `false`.

Example

In the following example, the condition inside the parentheses evaluates the variable `name` to see if it has the literal value `"Erica"`. If it does, the `play()` function inside the curly braces runs.

```
if(name == "Erica"){
    play();
}
```

The following example uses an `if` statement to evaluate how long it takes a user to click the `submit_btn` instance in a SWF file. If a user clicks the button more than 10 seconds after the SWF file plays, the condition evaluates to `true` and the message inside the curly braces ({}) appears in a text field that's created at runtime (using `createTextField()`). If the user clicks the button less than 10 seconds after the SWF file plays, the condition evaluates to `false` and a different message appears.

```
this.createTextField("message_txt", this.getNextHighestDepth, 0, 0, 100, 22);
message_txt.autoSize = true;
var startTime:Number = getTimer();
this.submit_btn.onRelease = function() {
    var difference:Number = (getTimer() - startTime) / 1000;
    if (difference > 10) {
    this._parent.message_txt.text = "Not very speedy, you took "+difference+" seconds.";
    }
    else {
    this._parent.message_txt.text = "Very good, you hit the button in "+difference+" seconds.";
    }
};
```

See also

`else statement`

implements statement

`className` `implements` *`interface01`* `[, interface02 , ...]`

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Specifies that a class must define all the methods declared in the interface (or interfaces) being implemented.

Availability

Flash Lite 2.0

Example

See `interface`.

See also

`class statement`

import statement

```
import className
import packageName.*
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This statement is supported in the Actions panel as well as in external class files.

Lets you access classes without specifying their fully qualified names. For example, if you want to use a custom class macr.util.users.UserClass in a script, you must refer to it by its fully qualified name or import it; if you import it, you can refer to it by the class name:

```
// before importing
var myUser:macr.util.users.UserClass = new macr.util.users.UserClass();
// after importing
import macr.util.users.UserClass;
var myUser:UserClass = new UserClass();
```

If there are several class files in the package (*working_directory*`/macr/utils/users`) that you want to access, you can import them all in a single statement, as shown in the following example:

```
import macr.util.users.*;
```

You must issue the `import` statement before you try to access the imported class without fully specifying its name.

If you import a class but don't use it in your script, the class isn't exported as part of the SWF file. This means you can import large packages without being concerned about the size of the SWF file; the bytecode associated with a class is included in a SWF file only if that class is actually used.

The `import` statement applies only to the current script (frame or object) in which it's called. For example, suppose on Frame 1 of a Flash document you import all the classes in the macr.util package. On that frame, you can reference classes in that package by their simple names:

```
// On Frame 1 of a FLA:
import macr.util.*;
var myFoo:foo = new foo();
```

On another frame script, however, you would need to reference classes in that package by their fully qualified names (`var myFoo:foo = new macr.util.foo();`) or add an `import` statement to the other frame that imports the classes in that package.

Availability
Flash Lite 2.0

Parameters
className : `String` - The fully qualified name of a class you have defined in an external class file.

interface statement

```
interface InterfaceName [extends InterfaceName ] {}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Defines an interface. An interface is similar to a class, with the following important differences:

- Interfaces contain only declarations of methods, not their implementation. That is, every class that implements an interface must provide an implementation for each method declared in the interface.
- Only public members are allowed in an interface definition; instance and class members are not permitted.
- The `get` and `set` statements are not allowed in interface definitions.

Example
The following example shows several ways to define and implement interfaces:

```
(in top-level package .as files Ia, B, C, Ib, D, Ic, E)
// filename Ia.as
interface Ia {
    function k():Number; // method declaration only
    function n(x:Number):Number; // without implementation
}
// filename B.as
class B implements Ia {
    function k():Number {
    return 25;
    }
    function n(x:Number):Number {
    return x + 5;
    }
} // external script or Actions panel // script file
var mvar:B = new B();
trace(mvar.k()); // 25
trace(mvar.n(7)); // 12
// filename c.as
class C implements Ia {
    function k():Number {
    return 25;
    }
} // error: class must implement all interface methods
// filename Ib.as
interface Ib {
    function o():Void;
}
class D implements Ia, Ib {
    function k():Number {
    return 15;
    }
    function n(x:Number):Number {
    return x * x;
    }
    function o():Void {
    trace("o");
    }
} // external script or Actions panel // script file
mvar = new D();
```

```
trace(mvar.k()); // 15
trace(mvar.n(7)); // 49
trace(mvar.o()); // "o"
interface Ic extends Ia {
    function p():Void;
}
class E implements Ib, Ic {
    function k():Number {
    return 25;
    }
    function n(x:Number):Number {
    return x + 5;
    }
    function o():Void {
    trace("o");
    }
    function p():Void {
    trace("p");
    }
}
```

See also

`class statement`

intrinsic statement

```
intrinsic class className [extends superClass] [implements interfaceName [, interfaceName...] ] {
    //class definition here
}
```

Allows compile-time type checking of previously defined classes. Flash uses intrinsic class declarations to enable compile-time type checking of built-in classes such as `Array`, `Object`, and `String`. This keyword indicates to the compiler that no function implementation is required, and that no bytecode should be generated for it.

The `intrinsic` keyword can also be used with variable and function declarations. Flash uses this keyword to enable compile-time type checking for global functions and properties.

The `intrinsic` keyword was created specifically to enable compile-time type checking for built-in classes and objects, and global variables and functions. This keyword was not meant for general purpose use, but may be of some value to developers seeking to enable compile-time type checking with previously defined classes, especially if the classes are defined using ActionScript 1.0.

This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Availability

Flash Lite 2.0

Example

The following example shows how to enable compile-time file checking for a previously defined ActionScript 1.0 class. The code will generate a compile-time error because the call `myCircle.setRadius()` sends a `String` value as a parameter instead of a `Number` value. You can avoid the error by changing the parameter to a `Number` value (for example, by changing `"10"` to `10`).

```
// The following code must be placed in a file named Circle.as
// that resides within your classpath:
intrinsic class Circle {
    var radius:Number;
    function Circle(radius:Number);
    function getArea():Number;
    function getDiameter():Number;
    function setRadius(param_radius:Number):Number;
}

// This ActionScript 1.0 class definition may be placed in your FLA file.
// Circle class is defined using ActionScript 1.0
function Circle(radius) {
    this.radius = radius;
    this.getArea = function(){
    return Math.PI*this.radius*this.radius;
    };
    this.getDiameter = function() {
    return 2*this.radius;
    };
    this.setRadius = function(param_radius) {
    this.radius = param_radius;
    }
}

// ActionScript 2.0 code that uses the Circle class
var myCircle:Circle = new Circle(5);
trace(myCircle.getArea());
trace(myCircle.getDiameter());
myCircle.setRadius("10");
trace(myCircle.radius);
trace(myCircle.getArea());
trace(myCircle.getDiameter());
```

See also

class statement

private statement

```
    class className{
    private var name;
    private function name() {
    // your statements here
    }
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Specifies that a variable or function is available only to the class that declares or defines it or to subclasses of that class. By default, a variable or function is available to any caller. Use this keyword if you want to restrict access to a variable or function.

You can use this keyword only in class definitions, not in interface definitions.

Availability
Flash Lite 2.0

Parameters
name : `String` - The name of the variable or function that you want to specify as private.

Example
The following example demonstrates how you can hide certain properties within a class using the `private` keyword. Create a new AS file called Login.as.

```
class Login {
    private var loginUserName:String;
    private var loginPassword:String;
    public function Login(param_username:String, param_password:String) {
    this.loginUserName = param_username;
    this.loginPassword = param_password;
    }
    public function get username():String {
    return this.loginUserName;
    }
    public function set username(param_username:String):Void {
    this.loginUserName = param_username;
    }
    public function set password(param_password:String):Void {
    this.loginPassword = param_password;
    }
}
```

In the same directory as Login.as, create a new FLA or AS document. Enter the following ActionScript in Frame 1 of the Timeline.

```
import Login;
var gus:Login = new Login("Gus", "Smith");
trace(gus.username); // output: Gus
trace(gus.password); // output: undefined
trace(gus.loginPassword); // error
```

Because `loginPassword` is a private variable, you cannot access it from outside the Login.as class file. Attempts to access the private variable generate an error message.

See also
`public statement`

public statement

```
class className{
    public var name;
    public function name() {
    // your statements here } }
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Specifies that a variable or function is available to any caller. Because variables and functions are public by default, this keyword is used primarily for stylistic reasons. For example, you might want to use it for reasons of consistency in a block of code that also contains private or static variables.

Availability

Flash Lite 2.0

Parameters

name: `String` - The name of the variable or function that you want to specify as public.

Example

The following example shows how you can use public variables in a class file. Create a new class file called User.as and enter the following code:

```
class User {
    public var age:Number;
    public var name:String;
}
```

Then create a new FLA or AS file in the same directory, and enter the following ActionScript in Frame 1 of the Timeline:

```
import User;
var jimmy:User = new User();
jimmy.age = 27;
jimmy.name = "jimmy";
```

If you change one of the public variables in the User class to a private variable, an error is generated when trying to access the property.

See also

`private statement`

return statement

`return[expression]`

Specifies the value returned by a function. The `return` statement evaluates `expression` and returns the result as a value of the function in which it executes. The `return` statement causes execution to return immediately to the calling function. If the `return` statement is used alone, it returns `undefined`.

You can't return multiple values. If you try to do so, only the last value is returned. In the following example, `c` is returned:

```
return a, b, c ;
```

If you need to return multiple values, you might want to use an array or object instead.

Availability

Flash Lite 2.0

Returns

`String` - The evaluated `expression` parameter, if provided.

Parameters

expression - A string, number, Boolean, array, or object to evaluate and return as a value of the function. This parameter is optional.

Example

The following example uses the `return` statement inside the body of the `sum()` function to return the added value of the three parameters. The next line of code calls `sum()` and assigns the returned value to the variable `newValue`.

```
function sum(a:Number, b:Number, c:Number):Number {
    return (a + b + c);
}
var newValue:Number = sum(4, 32, 78);
trace(newValue); // output: 114
```

See also

`Array function`

set statement

```
function set property(varName) {
    // your statements here
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Permits implicit setting of properties associated with objects based on classes you have defined in external class files. Using implicit set methods lets you modify the value of an object's property without accessing the property directly. Implicit get/set methods are syntactic shorthand for the `Object.addProperty()` method in ActionScript 1.

Availability

Flash Lite 2.0

Parameters

property : `String` - Word that refers to the property that `set` will access; this value must be the same as the value used in the corresponding `get` command.

Example

The following example creates a Login class that demonstrates how the `set` keyword can be used to set private variables:

```
class Login {
    private var loginUserName:String;
    private var loginPassword:String;
    public function Login(param_username:String, param_password:String) {
    this.loginUserName = param_username;
    this.loginPassword = param_password;
    }
    public function get username():String {
    return this.loginUserName;
    }
    public function set username(param_username:String):Void {
    this.loginUserName = param_username;
    }
    public function set password(param_password:String):Void {
    this.loginPassword = param_password;
    }
}
```

In a FLA or AS file that is in the same directory as Login.as, enter the following ActionScript in Frame 1 of the Timeline:

```
var gus:Login = new Login("Gus", "Smith");
trace(gus.username); // output: Gus
gus.username = "Rupert";
trace(gus.username); // output: Rupert
```

In this example, the `get` function executes when the value is traced. The `set` function triggers only when you pass it a value, as shown in the line:

```
gus.username = "Rupert";
```

See also

`getProperty function`

set variable statement

`set("variableString",expression)`

Assigns a value to a variable. A *variable* is a container that holds data. The container is always the same, but the contents can change. By changing the value of a variable as the SWF file plays, you can record and save information about what the user has done, record values that change as the SWF file plays, or evaluate whether a condition is `true` or `false`.

Variables can hold any data type (for example, String, Number, Boolean, Object, or MovieClip). The Timeline of each SWF file and movie clip has its own set of variables, and each variable has its own value independent of variables on other Timelines.

Strict data typing is not supported inside a `set` statement. If you use this statement to set a variable to a value whose data type is different from the data type associated with the variable in a class file, no compiler error is generated.

A subtle but important distinction to bear in mind is that the parameter `variableString` is a string, not a variable name. If you pass an existing variable name as the first parameter to `set()` without enclosing the name in quotation marks (""), the variable is evaluated before the value of `expression` is assigned to it. For example, if you create a string variable named `myVariable` and assign it the value "`Tuesday`", and then forget to use quotation marks, you will inadvertently create a new variable named `Tuesday` that contains the value you intended to assign to `myVariable`:

```
var myVariable:String = "Tuesday";
set (myVariable, "Saturday");
trace(myVariable); // outputs Tuesday
trace(Tuesday); // outputs Saturday
```

You can avoid this situation by using quotation marks (""):

```
set ("myVariable", "Saturday");
trace(myVariable); //outputs Saturday
```

Availability
Flash Lite 2.0

Parameters
variableString: `String` - A string that names a variable to hold the value of the *expression* parameter.

Example
In the following example, you assign a value to a variable. You are assigning the value of `"Jakob"` to the `name` variable.

```
set("name", "Jakob");
trace(name);
```

The following code loops three times and creates three new variables, called `caption0`, `caption1`, and `caption2`:

```
for (var i = 0; i < 3; i++) {
    set("caption" + i, "this is caption " + i);
}
trace(caption0);
trace(caption1);
trace(caption2);
```

static statement

```
class className{
    static var name;
    static function name() {
    // your statements here } }
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Specifies that a variable or function is created only once per class rather than being created in every object based on that class.

You can access a static class member without creating an instance of the class by using the syntax `someClassName.name`. If you do create an instance of the class, you can also access a static member using the instance, but only through a non-static function that accesses the static member.

You can use this keyword in class definitions only, not in interface definitions.

Availability
Flash Lite 2.0

Parameters
name: `String` - The name of the variable or function that you want to specify as static.

Example

The following example demonstrates how you can use the `static` keyword to create a counter that tracks how many instances of the class have been created. Because the `numInstances` variable is static, it will be created only once for the entire class, not for every single instance. Create a new AS file called Users.as and enter the following code:

```
class Users {
    private static var numInstances:Number = 0;
    function Users() {
    numInstances++;
    }
    static function get instances():Number {
    return numInstances;
    }
}
```

Create a FLA or AS document in the same directory, and enter the following ActionScript in Frame 1 of the Timeline:

```
trace(Users.instances);
var user1:Users = new Users();
trace(Users.instances);
var user2:Users = new Users();
trace(Users.instances);
```

See also

`private statement`

super statement

```
super.method([arg1, ..., argN])
super([arg1, ..., argN])
```

The first syntax style may be used within the body of an object method to invoke the superclass version of a method, and can optionally pass parameters `(arg1 ... argN)` to the superclass method. This is useful for creating subclass methods that add additional behavior to superclass methods, but also invoke the superclass methods to perform their original behavior.

The second syntax style may be used within the body of a constructor function to invoke the superclass version of the constructor function and may optionally pass it parameters. This is useful for creating a subclass that performs additional initialization, but also invokes the superclass constructor to perform superclass initialization.

Availability

Flash Lite 2.0

Returns

Both forms invoke a function. The function may return any value.

Parameters

method : `Function` - The method to invoke in the superclass.

argN - Optional parameters that are passed to the superclass version of the method (syntax 1) or to the constructor function of the superclass (syntax 2).

switch statement

`switch (expression){caseClause: [defaultClause:] }`

Creates a branching structure for ActionScript statements. As with the `if` statement, the `switch` statement tests a condition and executes statements if the condition returns a value of `true`. All switch statements should include a default case. The default case should include a break statement that prevents a fall-through error if another case is added later. When a case falls through, it doesn't have a break statement.

Availability

Flash Lite 1.0

Parameters

expression - Any expression.

Example

In the following example, if the `String.fromCharCode(Key.getAscii())` parameter evaluates to `A`, the `trace()` statement that follows `case "A"` executes; if the parameter evaluates to `a`, the `trace()` statement that follows `case "a"` executes; and so on. If no `case` expression matches the `String.fromCharCode(Key.getAscii())` parameter, the `trace()` statement that follows the `default` keyword executes.

```
var listenerObj:Object = new Object();
listenerObj.onKeyDown = function() {
    switch (String.fromCharCode(Key.getAscii())) {
    case "A" :
    trace("you pressed A");
    break;
    case "a" :
    trace("you pressed a");
    break;
    case "E" :
    case "e" :
    trace("you pressed E or e");
    break;
    case "I" :
    case "i" :
    trace("you pressed I or i");
    break;
    default :
    trace("you pressed some other key");
    break;
    }
};
Key.addListener(listenerObj);
```

See also

`===` strict equality operator

throw statement

`throw expression`

Generates, or *throws*, an error that can be handled, or *caught*, by a `catch{}` code block. If an exception is not caught by a `catch` block, the string representation of the thrown value is sent to the Output panel.

Typically, you throw instances of the Error class or its subclasses (see the Example section).

Availability

Flash Lite 2.0

Parameters

expression : `Object` - An ActionScript expression or object.

Example

In this example, a function named `checkEmail()` checks whether the string that is passed to it is a properly formatted e-mail address. If the string does not contain an @ symbol, the function throws an error.

```
function checkEmail(email:String) {
    if (email.indexOf("@") == -1) {
    throw new Error("Invalid email address");
    }
}
checkEmail("someuser_theirdomain.com");
```

The following code then calls the `checkEmail()` function within a `try` code block. If the `email_txt` string does not contain a valid e-mail address, the error message appears in a text field (`error_txt`).

```
try {
    checkEmail("Joe Smith");
}
catch (e) {
    error_txt.text = e.toString();
}
```

In the following example, a subclass of the Error class is thrown. The `checkEmail()` function is modified to throw an instance of that subclass.

```
// Define Error subclass InvalidEmailError // In InvalidEmailError.as: class
InvalidEmailAddress extends Error { var message = "Invalid email address."; }
```

In a FLA or AS file, enter the following ActionScript in Frame 1 of the Timeline:

```
import InvalidEmailAddress;
function checkEmail(email:String) {
    if (email.indexOf("@") == -1) {
    throw new InvalidEmailAddress();
    }
}
try {
    checkEmail("Joe Smith");
}
catch (e) {
    this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
    error_txt.autoSize = true;
    error_txt.text = e.toString();
}
```

See also

Error

try..catch..finally statement

```
try {// ... try block ... }
    finally { // ... finally block ... }
try { // ... try block ... }
catch(error [:ErrorType1]) // ... catch block ... }
[catch(error[:ErrorTypeN]) { // ... catch block ... }]
[finally { // ... finally block ... }]
```

Enclose a block of code in which an error can occur, and then respond to the error. If any code within the `try` code block throws an error (using the `throw` statement), control passes to the `catch` block, if one exists, and then to the `finally` code block, if one exists. The `finally` block always executes, regardless of whether an error was thrown. If code within the `try` block doesn't throw an error (that is, if the `try` block completes normally), then the code in the `finally` block is still executed. The `finally` block executes even if the `try` block exits using a `return` statement.

A `try` block must be followed by a `catch` block, a `finally` block, or both. A single `try` block can have multiple `catch` blocks but only one `finally` block. You can nest `try` blocks as many levels deep as desired.

The `error` parameter specified in a `catch` handler must be a simple identifier such as `e` or `theException` or `x`. The variable in a `catch` handler can also be typed. When used with multiple `catch` blocks, typed errors let you catch multiple types of errors thrown from a single `try` block.

If the exception thrown is an object, the type will match if the thrown object is a subclass of the specified type. If an error of a specific type is thrown, the `catch` block that handles the corresponding error is executed. If an exception that is not of the specified type is thrown, the `catch` block does not execute and the exception is automatically thrown out of the `try` block to a `catch` handler that matches it.

If an error is thrown within a function, and the function does not include a `catch` handler, then the ActionScript interpreter exits that function, as well as any caller functions, until a `catch` block is found. During this process, `finally` handlers are called at all levels.

Availability
Flash Lite 2.0

Parameters
error : `Object` - The expression thrown from a `throw` statement, typically an instance of the Error class or one of its subclasses.

Example
The following example shows how to create a `try..finally` statement. Because code in the `finally` block is guaranteed to execute, it is typically used to perform any necessary clean-up after a `try` block executes. In the following example, `setInterval()` calls a function every 1000 milliseconds (1 second). If an error occurs, an error is thrown and is caught by the `catch` block. The `finally` block is always executed whether or not an error occurs. Because `setInterval()` is used, `clearInterval()` must be placed in the `finally` block to ensure that the interval is cleared from memory.

```
myFunction = function () {
    trace("this is myFunction");
};
try {
    myInterval = setInterval(this, "myFunction", 1000);
    throw new Error("my error");
}
catch (myError:Error) {
    trace("error caught: "+myError);
}
finally {
    clearInterval(myInterval);
    trace("error is cleared");
}
```

In the following example, the `finally` block is used to delete an ActionScript object, regardless of whether an error occurred. Create a new AS file called Account.as.

```
class Account {
    var balance:Number = 1000;
    function getAccountInfo():Number {
    return (Math.round(Math.random() * 10) % 2);
    }
}
```

In the same directory as Account.as, create a new AS or FLA document and enter the following ActionScript in Frame 1 of the Timeline:

```
import Account;
var account:Account = new Account();
try {
    var returnVal = account.getAccountInfo();
    if (returnVal != 0) {
    throw new Error("Error getting account information.");
    }
}
finally {
    if (account != null) {
    delete account;
    }
}
```

The following example demonstrates a `try..catch` statement. The code within the `try` block is executed. If an exception is thrown by any code within the `try` block, control passes to the `catch` block, which shows the error message in a text field using the `Error.toString()` method.

In the same directory as Account.as, create a new FLA document and enter the following ActionScript in Frame 1 of the Timeline:

```
import Account;
var account:Account = new Account();
try {
    var returnVal = account.getAccountInfo();
    if (returnVal != 0) {
    throw new Error("Error getting account information.");
    }
    trace("success");
}
catch (e) {
    this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
    status_txt.autoSize = true;
    status_txt.text = e.toString();
}
```

The following example shows a `try` code block with multiple, typed `catch` code blocks. Depending on the type of error that occurred, the `try` code block throws a different type of object. In this case, `myRecordSet` is an instance of a (hypothetical) class named RecordSet whose `sortRows()` method can throw two types of errors, RecordSetException and MalformedRecord.

In the following example, the RecordSetException and MalformedRecord objects are subclasses of the Error class. Each is defined in its own AS class file.

```
// In RecordSetException.as:
class RecordSetException extends Error {
    var message = "Record set exception occurred.";
}
// In MalformedRecord.as:
class MalformedRecord extends Error {
    var message = "Malformed record exception occurred.";
}
```

Within the RecordSet class's `sortRows()` method, one of these previously defined error objects is thrown, depending on the type of exception that occurred. The following example shows how this code might look:

```
class RecordSet {
    function sortRows() {
    var returnVal:Number = randomNum();
    if (returnVal == 1) {
    throw new RecordSetException();
    }
    else if (returnVal == 2) {
    throw new MalformedRecord();
    }
    }
    function randomNum():Number {
    return Math.round(Math.random() * 10) % 3;
    }
}
```

Finally, in another AS file or FLA script, the following code invokes the `sortRows()` method on an instance of the RecordSet class. It defines `catch` blocks for each type of error that is thrown by `sortRows()`

```
import RecordSet;
var myRecordSet:RecordSet = new RecordSet();
try {
    myRecordSet.sortRows();
    trace("everything is fine");
}
catch (e:RecordSetException) {
    trace(e.toString());
}
catch (e:MalformedRecord) {
    trace(e.toString());
}
```

See also

Error

var statement

`var variableName [= value1][...,variableNameN[=valueN]]`

Used to declare local variables. If you declare variables inside a function, the variables are local. They are defined for the function and expire at the end of the function call. More specifically, a variable defined using `var` is local to the code block containing it. Code blocks are demarcated by curly braces ({}).

If you declare variables outside a function, the variables are available throughout the timeline containing the statement.

You cannot declare a variable scoped to another object as a local variable.

```
my_array.length = 25; // ok
var my_array.length = 25; // syntax error
```

When you use `var`, you can strictly type the variable.

You can declare multiple variables in one statement, separating the declarations with commas (although this syntax may reduce clarity in your code):

```
var first:String = "Bart", middle:String = "J.", last:String = "Bartleby";
```

Note: You must also use var when declaring properties inside class definitions in external scripts. Class files also support public, private, and static variable scopes.

Availability

Flash Lite 2.0

Parameters

variableName : `String` - An identifier.

Example

The following ActionScript creates a new array of product names. `Array.push` adds an element onto the end of the array. If you want to use strict typing, it is essential that you use the `var` keyword. Without `var` before `product_array`, you get errors when you try to use strict typing.

```
var product_array:Array = new Array("MX 2004", "Studio", "Dreamweaver", "Flash", "ColdFusion",
"Contribute", "Breeze");
product_array.push("Flex");
trace(product_array);
// output: MX 2004,Studio,Dreamweaver,Flash,ColdFusion,Contribute,Breeze,Flex
```

while statement

```
while(condition) { statement(s); }
```

Evaluates a condition and if the condition evaluates to `true`, runs a statement or series of statements before looping back to evaluate the condition again. After the condition evaluates to `false`, the statement or series of statements is skipped and the loop ends.

The `while` statement performs the following series of steps. Each repetition of steps 1 through 4 is called an *iteration* of the loop. The *condition* is retested at the beginning of each iteration, as shown in the following steps:

- The expression *condition* is evaluated.
- If *condition* evaluates to `true` or a value that converts to the Boolean value `true`, such as a nonzero number, go to step 3. Otherwise, the `while` statement is completed and execution resumes at the next statement after the `while` loop.
- Run the statement block *statement(s)*.
- Go to step 1.

Looping is commonly used to perform an action while a counter variable is less than a specified value. At the end of each loop, the counter is incremented until the specified value is reached. At that point, the *condition* is no longer `true`, and the loop ends.

The curly braces ({}) used to enclose the block of statements to be executed by the `while` statement are not necessary if only one statement will execute.

Availability

Flash Lite 1.0

Parameters

condition : `Boolean` - An expression that evaluates to `true` or `false`.

Example

In the following example, the `while` statement is used to test an expression. When the value of `i` is less than 20, the value of `i` is traced. When the condition is no longer `true`, the loop exits.

```
var i:Number = 0;
while (i < 20) {
    trace(i);
    i += 3;
}
```

The following result is displayed in the Output panel.

```
0
3
6
9
12
15
18
```

See also

`continue statement`

with statement

`with (object:Object) { statement(s); }`

Lets you specify an object (such as a movie clip) with the *object* parameter and evaluate expressions and actions inside that object with the *statement(s)* parameter. This prevents you from having to repeatedly write the object's name or the path to the object.

The *object* parameter becomes the context in which the properties, variables, and functions in the *statement(s)* parameter are read. For example, if *object* is `my_array`, and two of the properties specified are `length` and `concat`, those properties are automatically read as `my_array.length` and `my_array.concat`. In another example, if *object* is `state.california`, any actions or statements inside the `with` statement are called from inside the `california` instance.

To find the value of an identifier in the *statement(s)* parameter, ActionScript starts at the beginning of the scope chain specified by the *object* and searches for the identifier at each level of the scope chain, in a specific order.

The scope chain used by the `with` statement to resolve identifiers starts with the first item in the following list and continues to the last item:

- The object specified in the *object* parameter in the innermost `with` statement.
- The object specified in the *object* parameter in the outermost `with` statement.
- The Activation object. (A temporary object that is automatically created when a function is called that holds the local variables called in the function.)
- The movie clip that contains the currently executing script.
- The Global object (built-in objects such as Math and String).

To set a variable inside a `with` statement, you must have declared the variable outside the `with` statement, or you must enter the full path to the Timeline on which you want the variable to live. If you set a variable in a `with` statement without declaring it, the `with` statement will look for the value according to the scope chain. If the variable doesn't already exist, the new value will be set on the Timeline from which the `with` statement was called.

Instead of using `with()`, you can use direct paths. If you find that paths are long and cumbersome to type, you can create a local variable and store the path in the variable, which you can then reuse in your code, as shown in the following ActionScript:

```
var shortcut = this._parent._parent.name_txt; shortcut.text = "Hank"; shortcut.autoSize = true;
```

Availability

Flash Lite 2.0

Parameters

object : `Object` - An instance of an ActionScript object or movie clip.

Example

The following example sets the `_x` and `_y` properties of the `someOther_mc` instance, and then instructs `someOther_mc` to go to Frame 3 and stop.

```
with (someOther_mc) {
    _x = 50;
    _y = 100;
    gotoAndStop(3);
}
```

The following code snippet shows how to write the preceding code without using a `with` statement.

```
someOther_mc._x = 50;
someOther_mc._y = 100;
someOther_mc.gotoAndStop(3);
```

The `with` statement is useful for accessing multiple items in a scope chain list simultaneously. In the following example, the built-in `Math` object is placed at the front of the scope chain. Setting `Math` as a default object resolves the identifiers `cos`, `sin`, and `PI` to `Math.cos`, `Math.sin`, and `Math.PI`, respectively. The identifiers `a`, `x`, `y`, and `r` are not methods or properties of the `Math` object, but because they exist in the object activation scope of the function `polar()`, they resolve to the corresponding local variables.

```
function polar(r:Number):Void {
    var a:Number, x:Number, y:Number;
    with (Math) {
    a = PI * pow(r, 2);
    x = r * cos(PI);
    y = r * sin(PI / 2);
    }
    trace("area = " + a);
    trace("x = " + x);
    trace("y = " + y);
} polar(3);
```

The following result is displayed in the Output panel.

```
area = 28.2743338823081
x = -3
y = 3
```

fscommand2 commands

The following commands are available for the `fscommand2()` function. For a description of the `fscommand2()` function, see fscommand2 Function under "Global functions."

fscommand2 fommands

Command	Description
ExtendBacklightDuration	Extends the duration of a backlight for a specified period of time.
FullScreen	Sets the size of the display area to be used for rendering.
GetBatteryLevel	Returns the current battery level.
GetDevice	Sets a parameter that identifies the device on which Flash Lite is running.
GetDeviceID	Sets a parameter that represents the unique identifier of the device (for example, the serial number).
GetFreePlayerMemory	Returns the amount of heap memory, in kilobytes, currently available to Flash Lite.

Command	Description
GetMaxBatteryLevel	Returns the maximum battery level of the device.
GetMaxSignalLevel	Returns the maximum signal strength level as a numeric value.
GetMaxVolumeLevel	Returns the maximum volume level of the device as a numeric value.
GetNetworkConnectionName	Returns the name of the active or default network connection.
GetNetworkConnectStatus	Returns a value that indicates the current network connection status.
GetNetworkGeneration	Returns the generation of the current mobile wireless network (such as 2G or second generation of mobile wireless).
GetNetworkName	Sets a parameter to the name of the current network.
GetNetworkRequestStatus	Returns a value indicating the status of the most recent HTTP request.
GetNetworkStatus	Returns a value indicating the network status of the phone (that is, whether there is a network registered and whether the phone is currently roaming).
GetPlatform	Sets a parameter that identifies the current platform, which broadly describes the class of device.
GetPowerSource	Returns a value that indicates whether the power source is currently supplied from a battery or from an external power source.
GetSignalLevel	Returns the current signal strength as a numeric value.
GetSoftKeyLocation	Returns a value that indicates the location of soft keys on the device.
GetTotalPlayerMemory	Returns the total amount of heap memory, in kilobytes, allocated to Flash Lite.
GetVolumeLevel	Returns the current volume level of the device as a numeric value.
Quit	Causes the Flash Lite Player to stop playback and exit.
ResetSoftKeys	Resets the soft keys to their original settings.
SetFocusRectColor	Sets the color of the focus rectangle to any color.
SetInputTextType	Specifies the mode in which the input text field should be opened.
SetSoftKeys	Remaps the softkeys of a mobile device.
StartVibrate	Starts the phone's vibration feature.
StopVibrate	Stops the current vibration, if any.

ExtendBacklightDuration fscommand2 command

`ExtendBacklightDuration`

Extends the duration of a backlight for a specified period of time.

If the duration is greater than zero, this command specifies the amount of time in seconds (maximum of 60 seconds) that the backlight should be kept on. If the time elapses without an additional call to this command, the backlight behavior reverts to the default duration. If duration is zero, the backlight behavior immediately reverts to the default behavior.

Note: This feature is system dependent. For example, some systems limit the total duration that the backlight can be extended.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
ExtendBacklightDuration	`duration` The backlight duration, in seconds. Maximum value of 60 seconds.	-1: Not supported. 0: An error occurred, and the operation could not be completed. 1: Success.

Availability
Flash Lite 2.0

Example
The following example extends the duration of the backlight for 45 seconds:

```
status = FSCommand2("ExtendBacklightDuration", 45)
```

FullScreen fscommand2 command

`FullScreen`

Sets the size of the display area to be used for rendering.

The size can be a defined variable or a constant string value, with one of these values: `true` (full screen) or `false` (less than full screen). Any other value is treated as the value `false`.

Note: This command is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

Command	Parameters	Value Returned
FullScreen	size	-1: Not supported. 0: Supported.

Availability
Flash Lite 1.1

Example
The following example sets the size of the display area to the full screen:

```
status = fscommand2("FullScreen", true);
```

GetBatteryLevel fscommand2 command

`GetBatteryLevel`

Returns the current battery level. It is a numeric value that ranges from 0 to the maximum value returned by the `GetMaxBatteryLevel` variable.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetBatteryLevel	None.	-1: Not supported. Other numeric values: The current battery level.

Availability
Flash Lite 1.1

Example
The following example sets the `battLevel` variable to the current level of the battery:

```
battLevel = fscommand2("GetBatteryLevel");
```

GetDevice fscommand2 command

`GetDevice`

Sets a parameter that identifies the device on which Flash Lite is running. This identifier is typically the model name.

Command	Parameters	Value Returned
GetDevice	*device* String to receive the identifier of the device. It can be either the name of a variable or a string value that contains the name of a variable.	-1: Not supported. 0: Supported..

Availability
Flash Lite 1.1

Example
The following example assigns the device identifier to the `device` variable:

```
status = fscommand2("GetDevice", "device");
```

Some sample results and the devices they signify follow:

`D506i` A Mitsubishi 506i phone. `DFOMA1` A Mitsubishi FOMA1 phone. `F506i` A Fujitsu 506i phone. `FFOMA1` A Fujitsu FOMA1 phone. `N506i` An NEC 506i phone. `NFOMA1` An NEC FOMA1 phone. `Nokia3650` A Nokia 3650 phone. `p506i` A Panasonic 506i phone. `PFOMA1` A Panasonic FOMA1 phone. `SH506i` A Sharp 506i phone. `SHFOMA1` A Sharp FOMA1 phone. `SO506i` A Sony 506iphone.

GetDeviceID fscommand2 command

`GetDeviceID`

Sets a parameter that represents the unique identifier of the device (for example, the serial number).

Command	Parameters	Value Returned
GetDeviceID	`id` A string to receive the unique identifier of the device. It can be either the name of a variable or a string value that contains the name of a variable.	-1: Not supported. 0: Supported.

Availability
Flash Lite 1.1

Example

The following example assigns the unique identifier to the `deviceID` variable:

```
status = fscommand2("GetDeviceID", "deviceID");
```

GetFreePlayerMemory fscommand2 command

`GetFreePlayerMemory`

Returns the amount of heap memory, in kilobytes, currently available to Flash Lite.

Command	Parameters	Value Returned
GetFreePlayerMemory	None	-1: Not supported. 0 or positive value: Available kilobytes of heap memory.

Availability

Flash Lite 1.1

Example

The following example sets `status` equal to the amount of free memory:

```
status = fscommand2("GetFreePlayerMemory");
```

GetMaxBatteryLevel fscommand2 command

`GetMaxBatteryLevel`

Returns the maximum battery level of the device. It is a numeric value greater than 0.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetMaxBatteryLevel	None	-1: Not supported. Other values: The maximum battery level.

Availability

Flash Lite 1.1

Example

The following example sets the `maxBatt` variable to the maximum battery level:

```
maxBatt = fscommand2("GetMaxBatteryLevel");
```

GetMaxSignalLevel fscommand2 command

`GetMaxSignalLevel`

Returns the maximum signal strength level as a numeric value.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetMaxSignalLevel	None	-1: Not supported. Other numeric values: The maximum signal level.

Availability
Flash Lite 1.1

Example
The following example assigns the maximum signal strength to the `sigStrengthMax` variable:

```
sigStrengthMax = fscommand2("GetMaxSignalLevel");
```

GetMaxVolumeLevel fscommand2 command

`GetMaxVolumeLevel`

Returns the maximum volume level of the device as a numeric value.

Command	Parameters	Value Returned
GetMaxVolumeLevel	None	-1: Not supported. Other values: The maximum volume level.

Availability
Flash Lite 1.1

Example
The following example sets the `maxvolume` variable to the maximum volume level of the device:

```
maxvolume = fscommand2("GetMaxVolumeLevel");
trace (maxvolume); // output: 80
```

GetNetworkConnectionName fscommand2 command

`GetNetworkConnectionName`

Returns the name of the active or default network connection. For mobile devices, this connection is also known as an access point.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetNetworkConnectionName	None	`-1`: Not supported. `0`: Success: returns the active network connection name. `1`: Success: returns the default network connection name. `2`: Unable to retrieve the connection name.

Availability
Flash Lite 2.0

Example

The following example returns the name of the active or default network connection in the argument `myConnectionName`:

```
status = FSCommand2("GetNetworkConnectionName", "myConnectionName");
```

GetNetworkConnectStatus fscommand2 command

`GetNetworkConnectStatus`

Returns a value that indicates the current network connection status.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetNetworkConnectStatus	None	`-1`: Not supported. `0`: There is currently an active network connection. `1`: The device is attempting to connect to the network. `2`: There is currently no active network connection. `3`: The network connection is suspended. `4`: The network connection cannot be determined.

Availability

Flash Lite 1.1

Example

The following example assigns the network connection status to the `connectstatus` variable, and then uses a `switch` statement to update a text field with the status of the connection:

```
connectstatus = FSCommand2("GetNetworkConnectStatus");
switch (connectstatus) {
    case -1 :
    _root.myText += "connectstatus not supported" + "\n";
    break;
    case 0 :
    _root.myText += "connectstatus shows active connection" + "\n";
    break;
    case 1 :
    _root.myText += "connectstatus shows attempting connection" + "\n";
    break;
    case 2 :
    _root.myText += "connectstatus shows no connection" + "\n";
    break;
    case 3 :
    _root.myText += "connectstatus shows suspended connection" + "\n";
    break;
    case 4 :
    _root.myText += "connectstatus shows indeterminable state" + "\n";
    break;
}
```

GetNetworkGeneration fscommand2 command

`GetNetworkGeneration`

Returns the generation of the current mobile wireless network, such as 2G (second generation of mobile wireless).

Command	Parameters	Value Returned
GetNetworkGeneration	None	`-1`: Not supported `0`: Unknown generation of mobile wireless network `1`: 2G `2`: 2.5G `3`: 3G

Availability
Flash Lite 2.0

Example
The following example shows syntax for returning the generation of the network:

```
status = fscommand2("GetNetworkGeneration");
```

GetNetworkName fscommand2 command

`GetNetworkName`

Sets a parameter to the name of the current network.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetNetworkName	`networkName` String representing the network name. It can be either the name of a variable or a string value that contains the name of a variable. If the network is registered and its name can be determined, `networkname` is set to the network name; otherwise, it is set to the empty string.	-1: Not supported. `0`: No network is registered. `1`: Network is registered, but network name is not known. `2`: Network is registered, and network name is known.

Availability
Flash Lite 1.1

Example
The following example assigns the name of the current network to the `myNetName` parameter and a status value to the `netNameStatus` variable:

```
netNameStatus = fscommand2("GetNetworkName", "myNetName");
```

GetNetworkRequestStatus fscommand2 command

`GetNetworkRequestStatus`

Returns a value indicating the status of the most recent HTTP request.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetNetworkRequestStatus	None	`-1`: The command is not supported. `0`: There is a pending request, a network connection has been established, the server's host name has been resolved, and a connection to the server has been made. `1`: There is a pending request, and a network connection is being established. `2`: There is a pending request, but a network connection has not yet been established. `3`: There is a pending request, a network connection has been established, and the server's host name is being resolved. `4`: The request failed because of a network error. `5`: The request failed because of a failure in connecting to the server. `6`: The server has returned an HTTP error (for example, 404). `7`: The request failed because of a failure in accessing the DNS server or in resolving the server name. `8`: The request has been successfully fulfilled. `9`: The request failed because of a timeout. `10`: The request has not yet been made.

Availability

Flash Lite 1.1

Example

The following example assigns the status of the most recent HTTP request to the `requeststatus` variable, and then uses a `switch` statement to update a text field with the status:

```
requeststatus = fscommand2("GetNetworkRequestStatus");
switch (requeststatus) {
    case -1:
    _root.myText += "requeststatus not supported" + "\n";
    break;
    case 0:
    _root.myText += "connection to server has been made" + "\n";
    break;
    case 1:
    _root.myText += "connection is being established" + "\n";
    break;
    case 2:
    _root.myText += "pending request, contacting network" + "\n";
    break;
    case 3:
    _root.myText += "pending request, resolving domain" + "\n";
    break;
    case 4:
    _root.myText += "failed, network error" + "\n";
    break;
    case 5:
    _root.myText += "failed, couldn't reach server" + "\n";
    break;
    case 6:
    _root.myText += "HTTP error" + "\n";
    break;
    case 7:
    _root.myText += "DNS failure" + "\n";
    break;
    case 8:
    _root.myText += "request has been fulfilled" + "\n";
    break;
    case 9:
    _root.myText += "request timedout" + "\n";
    break;
    case 10:
    _root.myText += "no HTTP request has been made" + "\n";
    break;
}
```

GetNetworkStatus fscommand2 command

`GetNetworkStatus`

Returns a value indicating the network status of the phone (that is, whether there is a network registered and whether the phone is currently roaming).

Command	Parameters	Value Returned
GetNetworkStatus	None	`-1`: The command is not supported. `0`: No network registered. `1`: On home network. `2`: On extended home network. `3`: Roaming (away from home network).

Availability
Flash Lite 1.1

Example
The following example assigns the status of the network connection to the `networkstatus` variable, and then uses a `switch` statement to update a text field with the status:

```
networkstatus = fscommand2("GetNetworkStatus");
switch(networkstatus) {
    case -1:
    _root.myText += "network status not supported" + "\n";
    break;
    case 0:
    _root.myText += "no network registered" + "\n";
    break;
    case 1:
    _root.myText += "on home network" + "\n";
    break;
    case 2:
    _root.myText += "on extended home network" + "\n";
    break;
    case 3:
    _root.myText += "roaming" + "\n";
    break;
}
```

GetPlatform fscommand2 command

`GetPlatform`

Sets a parameter that identifies the current platform, which broadly describes the class of device. For devices with open operating systems, this identifier is typically the name and version of the operating system.

Command	Parameters	Value Returned
GetPlatform	`platform` String to receive the identifier of the platform.	`-1`: Not supported. `0`: Supported.

Availability
Flash Lite 1.1

Example
The following example sets the `platform` parameter to the identifier for the current platform:

```
status = fscommand2("GetPlatform", "platform");
```

The following examples show some sample results for `platform`:

`506i` A 506i phone. `FOMA1` A FOMA1 phone. `Symbian6.1_s60.1` A Symbian 6.1, Series 60 version 1 phone. `Symbian7.0` A Symbian 7.0 phone

GetPowerSource fscommand2 command

`GetPowerSource`

Returns a value that indicates whether the power source is currently supplied from a battery or from an external power source.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetPowerSource	None	`-1`: Not supported. `0`: Device is operating on battery power. `1`: Device is operating on an external power source.

Availability
Flash Lite 1.1

Example
The following example sets the `myPower` variable to indicate the power source, or to `-1` if it was unable to do so:

```
myPower = fscommand2("GetPowerSource");
```

GetSignalLevel fscommand2 command

`GetSignalLevel`

Returns the current signal strength as a numeric value.

Note: This command is not supported for BREW devices.

Command	Parameters	Value Returned
GetSignalLevel	None	`-1`: Not supported. Other numeric values: The current signal level, ranging from `0` to the maximum value returned by `GetMaxSignalLevel`.

Availability
Flash Lite 1.1

Example
The following example assigns the signal level value to the `sigLevel` variable:

```
sigLevel = fscommand2("GetSignalLevel");
```

GetSoftKeyLocation fscommand2 command

`GetSoftKeyLocation`

Returns a value that indicates the location of soft keys on the device.

Command	Parameters	Value Returned
GetSoftKeyLocation	None	-1: Not supported. 0: Soft keys on top. 1: Soft keys on left. 2: Soft keys on bottom. 3: Soft keys on right.

Availability
Flash Lite 2.0

Example
The following example sets the `status` variable to indicate the soft key location, or to `-1` if soft keys are not supported on the device:

```
status = fscommand2("GetSoftKeyLocation");
```

GetTotalPlayerMemory fscommand2 command

`GetTotalPlayerMemory`

Returns the total amount of heap memory, in kilobytes, allocated to Flash Lite.

Command	Parameters	Value Returned
GetTotalPlayerMemory	None	`-1`: Not supported. `0` or positive value: Total kilobytes of heap memory.

Availability
Flash Lite 1.1

Example
The following example sets the `status` variable to the total amount of heap memory:

```
status = fscommand2("GetTotalPlayerMemory");
```

GetVolumeLevel fscommand2 command

`GetVolumeLevel`

Returns the current volume level of the device as a numeric value.

Command	Parameters	Value Returned
GetVolumeLevel	None	`-1`: Not supported. Other numeric values: The current volume level, ranging from `0` to the value returned by `fscommand2("GetMaxVolumeLevel")`.

Availability
Flash Lite 1.1

Example
The following example assigns the current volume level to the volume variable:

```
volume = fscommand2("GetVolumeLevel");
trace (volume); // output: 50
```

Quit fscommand2 command

`Quit`

Causes the Flash Lite player to stop playback and exit.

This command is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

Command	Parameters	Value Returned
Quit	None	`-1`: Not supported.

Availability
Flash Lite 1.1

Example
The following example causes Flash Lite to stop playback and quit when running in stand-alone mode:

```
status = fscommand2("Quit");
```

ResetSoftKeys fscommand2 command

`ResetSoftKeys`

Resets the soft keys to their original settings.

This command is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

Command	Parameters	Value Returned
ResetSoftKeys	None	`-1`: Not supported.

Availability
Flash Lite 1.1

Example
The following statement resets the soft keys to their original settings:

```
status = fscommand2("ResetSoftKeys");
```

SetFocusRectColor fscommand2 command

`SetFocusRectColor`

Sets the color of the focus rectangle to any color.

The acceptable range of values for red, green, and blue is 0-255. For Flash, you cannot change the default color of the focus rectangle, which is yellow.

Command	Parameters	Value Returned
SetFocusRectColor	None	`-1`: Not supported. `0`: Indeterminable. `1`: Success.

Availability
Flash Lite 2.0

Example
The following statement resets the color of the focus rectangle:

```
status = fscommand2("SetFocusRectColor, <red>, <green>, <blue>);
```

SetInputTextType fscommand2 command

`SetInputTextType`

Specifies the mode in which the input text field should be opened.

Flash Lite supports input text functionality by asking the host application to start the generic device-specific text input interface, often referred to as the front-end processor (FEP). When the `SetInputTextType` command is not used, the FEP is opened in default mode.

Command	Parameters	Value Returned
SetInputTextType	`variableName` Name of the input text field. It can be either the name of a variable or a string value that contains the name of a variable. **Note:** A text field's variable name is not the same as its instance name. You can specify a text field's variable name in the Var text box in the Property inspector or by using ActionScript. For example, the following code restricts input to numeric characters for the text field instance (`numTxt`) whose associated variable name is "`numTxt_var`". `var numTxt:TextField;numTxt.variable = "numTxt_var";fscommand2("SetInputTextType", "numTxt_var", "Numeric");` `type` One of the values `Numeric, Alpha`, `Alphanumeric`, `Latin`, `NonLatin`, or `NoRestriction`.	`0`: Failure. `1`: Success.

The following table shows what effect each mode has, and what modes are substituted:

InputTextType Mode	Sets the FEP to one of these mutually exclusive modes	If not supported on current device, opens the FEP in this mode
Numeric	Numbers only (0 to 9)	Alphanumeric
Alpha	Alphabetic characters only (A to Z, a to z)	Alphanumeric
Alphanumeric	Alphanumeric characters only (0 to 9, A to Z, a to z)	Latin
Latin	Latin characters only (alphanumeric and punctuation)	NoRestriction

InputTextType Mode	Sets the FEP to one of these mutually exclusive modes	If not supported on current device, opens the FEP in this mode
NonLatin	Non-Latin characters only (for example, Kanji and Kana)	NoRestriction
NoRestriction	Default mode (sets no restriction on the FEP)	N/A
NOTE: Not all mobile phones support these input text field types. For this reason, you must validate the input text data.		

Availability

Flash Lite 1.1

Example

The following line of code sets the input text type of the field associated with the `input1` variable to receive numeric data:

```
status = fscommand2("SetInputTextType", "input1", "Numeric");
```

SetSoftKeys fscommand2 command

`SetSoftKeys`

Remaps the soft keys of a mobile device.

When the user presses a soft key, any ActionScript associated with the soft key event is executed. The Flash Lite player executes this function immediately upon invocation. This command is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

For backward compatibility with Flash Lite 1.1, the `SOFT1` soft key is always mapped to the left key on the handset, and the `SOFT2` soft key is always mapped to the right key on the handset. For the `SOFT3` soft key and higher, the locations are dependent on each handset.

The arguments for this command specify the text to be displayed for the corresponding soft keys. When the `SetSoftKeys` command is executed, pressing the left key generates a `SOFT1` keypress event, and pressing the right key generates a `SOFT2` keypress event. Pressing the `SOFT3` through `SOFT12` soft keys generates their respective events.

Note: The remapping of soft keys depends on the mobile device. Check with the device manufacturer to see if the remapping of soft keys is supported.

Command	Parameters	Value Returned
SetSoftKeys	`soft1` Text to be displayed for the `SOFT1` soft key. `soft2` Text to be displayed for the `SOFT2` soft key. These parameters are either names of variables or constant string values (for example, "`Previous`").	-1: Not supported. 0: Supported.

Availability

Flash Lite 1.1

Example

The following example labels the `SOFT1` soft key "Previous" and the `SOFT2` soft key "Next":

```
status = fscommand2("SetSoftKeys", "Previous", "Next");
```

You can define variables or use constant string values for each soft key:

```
status = fscommand2("SetSoftKeys", soft1, soft2, [soft3], [soft4], ..., [softn])
```

Note: You can set one soft key without setting the others. These examples show the syntax and behavior of setting a specific soft key without affecting other keys:

- To set the left soft key label to "soft1" and the right soft key to empty:

  ```
  status = fscommand2("SetSoftKeys", "soft1", "")
  ```

- To leave the label for the left soft key as is and set right soft key to "soft2":

  ```
  status = fscommand2("SetSoftKeys", undefined, "soft2")
  ```

- To leave the label for the left soft key as is and set the right soft key to "soft2":

  ```
  status = fscommand2("SetSoftKeys", null, "soft2")
  ```

- To set the left soft key label to "soft1" and leave the right soft key as is:

  ```
  status = fscommand2("SetSoftKeys", "soft1")
  ```

StartVibrate fscommand2 command

`StartVibrate`

Starts the phone's vibration feature.

If a vibration is already occurring, Flash Lite stops that vibration before starting the new one. Vibrations also stop when playback of the Flash application is stopped or paused, and when the Flash Lite player quits.

Command	Parameters	Value Returned
StartVibrate	`time_on` Amount of time, in milliseconds (to a maximum of 5 seconds), that the vibration is on. `time_off` Amount of time, in milliseconds (to a maximum of 5 seconds), that the vibration is off. `repeat` Number of times (to a maximum of 3) to repeat this vibration.	`-1`: Not supported. `0`: Vibration was started. `1`: An error occurred and vibration could not be started.

Availability

Flash Lite 1.1

Example

The following example attempts to start a vibration sequence of 2.5 seconds on, 1 second off, repeated twice. It assigns a value to the `status` variable that indicates success or failure:

```
fscommand2("StartVibrate", 2500, 1000, 2);
```

StopVibrate fscommand2 command

`StopVibrate`

Stops the current vibration, if any.

Command	Parameters	Value Returned
StopVibrate	None	`-1`: Not supported. `0`: The vibration stopped.

Availability

Flash Lite 1.1

Example

The following example calls `StopVibrate` and saves the result (not supported or vibration stopped) in the `status` variable:

```
status = fscommand2("StopVibrate");
```

Chapter 2: ActionScript classes

Documentation for ActionScript classes includes syntax, usage information, and code samples for methods, properties, and events that belong to specific classes. The classes are listed alphabetically. If you are not sure to which class a method, property, or event belongs, search the Index.

arguments

```
Object
    |
    +-arguments
```

```
public class arguments
extends Object
```

An arguments object is used to store and access a function's arguments. While inside the function's body it can be accessed with the local `arguments` variable.

The arguments are stored as array elements, the first is accessed as `arguments[0]`, the second as `arguments[1]`, etc. The `arguments.length` property indicates the number of arguments passed to the function. Note that there may be a different number of arguments passed in than the function declares.

Availability

Flash Lite 2.0

See also

`Function`

Property summary

Modifiers	Property	Description
	callee : Object	A reference to the currently executing function.
	caller : Object	A reference to the function that called the currently executing function, or `null` if it wasn't called from another function.
	length : Number	The number of arguments passed to the function.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property) prototype (Object.prototype property), __resolve (Object.__resolve property)

Method summary

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

callee (arguments.callee property)

```
public callee : Object
```

A reference to the currently executing function.

Availability

Flash Lite 2.0

See also

caller (arguments.caller property)

caller (arguments.caller property)

```
public caller : Object
```

A reference to the function that called the currently executing function, or `null` if it wasn't called from another function.

Availability

Flash Lite 2.0

See also

callee (arguments.callee property)

length (arguments.length property)

```
public length : Number
```

The number of arguments passed to the function. This may be more or less than the function declares.

Availability

Flash Lite 2.0

Array

```
Object
    |
    +-Array
```

```
public dynamic class Array
extends Object
```

The Array class lets you access and manipulate indexed arrays. An indexed array is an object whose properties are identified by a number representing their position in the array. This number is referred to as the *index*. All indexed arrays are zero-based, which means that the first element in the array is [0], the second element is [1], and so on. To create an Array object, you use the constructor `newArray()`. To access the elements of an array, you use the array access (`[]`) operator.

You can store a wide variety of data types in an array element, including numbers, strings, objects, and even other arrays. You can create a *multidimensional* array by creating an indexed array and assigning to each of its elements a different indexed array. Such an array is considered multidimensional because it can be used to represent data in a table.

Array assignment is by reference rather than by value: when you assign one array variable to another array variable, both refer to the same array:

```
var oneArray:Array = new Array("a", "b", "c");
var twoArray:Array = oneArray; // Both array variables refer to the same array.
twoArray[0] = "z";
trace(oneArray); // Output: z,b,c.
```

The Array class should not be used to create *associative arrays*, which are different data structures that contain named elements instead of numbered elements. You should use the Object class to create associative arrays (also called *hashes*). Although ActionScript permits you to create associative arrays using the Array class, you cannot use any of the Array class methods or properties. At its core, an associative array is an instance of the Object class, and each key-value pair is represented by a property and its value. Another reason to declare an associative array as a type Object is that you can then use an object literal to populate your associative array (but only at the time you declare it). The following example creates an associative array using an object literal, accesses items using both the dot operator and the array access operator, and then adds a new key-value pair by creating a new property:

```
var myAssocArray:Object = {fname:"John", lname:"Public"};
trace(myAssocArray.fname); // Output: John
trace(myAssocArray["lname"]); // Output: Public
myAssocArray.initial = "Q";
trace(myAssocArray.initial); // Output: Q
```

Availability

Flash Lite 2.0

Example

In the following example, `my_array` contains four months of the year:

```
var my_array:Array = new Array();
my_array[0] = "January";
my_array[1] = "February";
my_array[2] = "March";
my_array[3] = "April";
```

Property summary

Modifiers	Property	Description
static	CASEINSENSITIVE : Number	Represents case-insensitive sorting.
static	DESCENDING : Number	Represents a descending sort order.
	length : Number	A non-negative integer specifying the number of elements in the array.
static	NUMERIC : Number	Represents numeric sorting instead of string-based sorting.
static	RETURNINDEXEDARRAY : Number	Represents the option to return an indexed array as a result of calling the `sort()` or `sortOn()` method.
static	UNIQUESORT : Number	Represents the unique sorting requirement.

Properties inherited from class Object

```
constructor (Object.constructor property), __proto__ (Object.__proto__
property) prototype (Object.prototype property), __resolve (Object.__resolve property)
```

Constructor summary

Signature	Description
Array ([value : Object])	Lets you create an array.

Method summary

Modifiers	Signature	Description
	concat ([value : Object]) : Array	Concatenates the elements specified in the parameters with the elements in an array and creates a new array.
	join ([delimiter : String]) : String	Converts the elements in an array to strings, inserts the specified separator between the elements, concatenates them, and returns the resulting string.
	pop () : Object	Removes the last element from an array and returns the value of that element.
	push (value : Object) : Number	Adds one or more elements to the end of an array and returns the new length of the array.
	reverse () : Void	Reverses the array in place.
	shift () : Object	Removes the first element from an array and returns that element.
	slice ([startIndex : Number] , [endIndex : Number]) : Array	Returns a new array that consists of a range of elements from the original array, without modifying the original array.
	sort ([compareFunction : Object] , [options : Number]) : Array	Sorts the elements in an array.

Modifiers	Signature	Description
	sortOn(fieldName:Object, [options:Object]) : Array	Sorts the elements in an array according to one or more fields in the array.
	splice(startIndex:Number, [deleteCount:Number], [value:Object]) : Array	Adds elements to and removes elements from an array.
	toString() : String	Returns a string value representing the elements in the specified Array object.
	unshift(value:Object) : Number	Adds one or more elements to the beginning of an array and returns the new length of the array.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

Array constructor

```
public Array([value:Object])
```

Lets you create an array. You can use the constructor to create different types of arrays: an empty array, an array with a specific length but whose elements have undefined values, or an array whose elements have specific values.

Usage 1: If you don't specify any parameters, an array with a length of 0 is created.

Usage 2: If you specify only a length, an array is created with `length` number of elements. The value of each element is set to `undefined`.

Usage 3: If you use the `element` parameters to specify values, an array is created with specific values.

Availability

Flash Lite 2.0

Parameters

value : Object [optional] - Either:

- An integer that specifies the number of elements in the array.
- A list of two or more arbitrary values. The values can be of type Boolean, Number, String, Object, or Array. The first element in an array always has an index or position of 0.

Note: If only a single numeric parameter is passed to the Array constructor, it is assumed to be `length` and it is converted to an integer by using the `Integer()` function.

Example

Usage 1: The following example creates a new Array object with an initial length of 0:

```
var my_array:Array = new Array();
trace(my_array.length); // Traces 0.
```

Usage 2: The following example creates a new Array object with an initial length of 4:

```
var my_array:Array = new Array(4);
trace(my_array.length); // Returns 4.
trace(my_array[0]); // Returns undefined.
if (my_array[0] == undefined) { // No quotation marks around undefined.
    trace("undefined is a special value, not a string");
} // Traces: undefined is a special value, not a string.
```

Usage 3: The following example creates the new Array object `go_gos_array` with an initial length of 5:

```
var go_gos_array:Array = new Array("Belinda", "Gina", "Kathy", "Charlotte", "Jane");
trace(go_gos_array.length); // Returns 5.
trace(go_gos_array.join(", ")); // Displays elements.
```

The initial elements of the `go_gos_array` array are identified, as shown in the following example:

```
go_gos_array[0] = "Belinda";
go_gos_array[1] = "Gina";
go_gos_array[2] = "Kathy";
go_gos_array[3] = "Charlotte";
go_gos_array[4] = "Jane";
```

The following code adds a sixth element to the `go_gos_array` array and changes the second element:

```
go_gos_array[5] = "Donna";
go_gos_array[1] = "Nina"
trace(go_gos_array.join(" + "));
// Returns Belinda + Nina + Kathy + Charlotte + Jane + Donna.
```

See also

`[] array access operator`, `length (Array.length property)`

CASEINSENSITIVE (Array.CASEINSENSITIVE property)

```
public static CASEINSENSITIVE : Number
```

Represents case-insensitive sorting. You can use this constant for the `options` parameter in the `sort()` or `sortOn()` method. The value of this constant is 1.

Availability

Flash Lite 2.0

See also

`sort (Array.sort method)`, `sortOn (Array.sortOn method)`

concat (Array.concat method)

```
public concat([value:Object]) : Array
```

Concatenates the elements specified in the parameters with the elements in an array and creates a new array. If the `value` parameters specify an array, the elements of that array are concatenated, rather than the array itself. The array `my_array` is left unchanged.

Availability

Flash Lite 2.0

Parameters

value : Object [optional] - Numbers, elements, or strings to be concatenated in a new array. If you don't pass any values, a duplicate of `my_array` is created.

Returns

Array - An array that contains the elements from this array followed by elements from the parameters.

Example

The following code concatenates two arrays:

```
var alpha_array:Array = new Array("a","b","c");
var numeric_array:Array = new Array(1,2,3);
var alphaNumeric_array:Array =alpha_array.concat(numeric_array);
trace(alphaNumeric_array);
// Creates array [a,b,c,1,2,3].
```

The following code concatenates three arrays:

```
var num1_array:Array = [1,3,5];
var num2_array:Array = [2,4,6];
var num3_array:Array = [7,8,9];
var nums_array:Array=num1_array.concat(num2_array,num3_array)
trace(nums_array);
// Creates array [1,3,5,2,4,6,7,8,9].
```

Nested arrays are not flattened in the same way as normal arrays. The elements in a nested array are not broken into separate elements in array `x_array`, as shown in the following example:

```
var a_array:Array = new Array ("a","b","c");

// 2 and 3 are elements in a nested array.
var n_array:Array = new Array(1, [2, 3], 4);

var x_array:Array = a_array.concat(n_array);
trace(x_array[0]); // a
trace(x_array[1]); // b
trace(x_array[2]); // c
trace(x_array[3]); // 1
trace(x_array[4]); // 2, 3
trace(x_array[5]); // 4
```

DESCENDING (Array.DESCENDING property)

```
public static DESCENDING : Number
```

Represents a descending sort order. You can use this constant for the `options` parameter in the `sort()` or `sortOn()` method. The value of this constant is 2.

Availability

Flash Lite 2.0

See also

`sort (Array.sort method)`, `sortOn (Array.sortOn method)`

join (Array.join method)

```
public join([delimiter:String]) : String
```

Converts the elements in an array to strings, inserts the specified separator between the elements, concatenates them, and returns the resulting string. A nested array is always separated by a comma (,), not by the separator passed to the `join()` method.

Availability

Flash Lite 2.0

Parameters

delimiter : String [optional] - A character or string that separates array elements in the returned string. If you omit this parameter, a comma (,) is used as the default separator.

Returns

String - A string.

Example

The following example creates an array with three elements: Earth, Moon, and Sun. It then joins the array three times—first by using the default separator (a comma [,] and a space), then by using a dash (-), and then by using a plus sign (+).

```
var a_array:Array = new Array("Earth","Moon","Sun")
trace(a_array.join());
// Displays Earth,Moon,Sun.
trace(a_array.join(" - "));
// Displays Earth - Moon - Sun.
trace(a_array.join(" + "));
// Displays Earth + Moon + Sun.
```

The following example creates a nested array that contains two arrays. The first array has three elements: Europa, Io, and Callisto. The second array has two elements: Titan and Rhea. It joins the array by using a plus sign (+), but the elements within each nested array remain separated by commas (,).

```
var a_nested_array:Array = new Array(["Europa", "Io", "Callisto"], ["Titan", "Rhea"]);
trace(a_nested_array.join(" + "));
// Returns Europa,Io,Callisto + Titan,Rhea.
```

See also

split (String.split method)

length (Array.length property)

```
public length : Number
```

A non-negative integer specifying the number of elements in the array. This property is automatically updated when new elements are added to the array. When you assign a value to an array element (for example, `my_array[index] = value`), if `index` is a number, and `index+1` is greater than the `length` property, the `length` property is updated to `index+1`.

Note: If you assign a value to the `length` property that is shorter than the existing length, the array will be truncated.

Availability
Flash Lite 2.0

Example
The following code explains how the `length` property is updated. The initial length is 0, and then updated to 1, 2, and 10. If you assign a value to the `length` property that is shorter than the existing length, the array will be truncated:

```
var my_array:Array = new Array();
trace(my_array.length); // initial length is 0
my_array[0] = "a";
trace(my_array.length); // my_array.length is updated to 1
my_array[1] = "b";
trace(my_array.length); // my_array.length is updated to 2
my_array[9] = "c";
trace(my_array.length); // my_array.length is updated to 10
trace(my_array);
// displays:
// a,b,undefined,undefined,undefined,undefined,undefined,undefined,undefined,c

// if the length property is now set to 5, the array will be truncated
my_array.length = 5;
trace(my_array.length); // my_array.length is updated to 5
trace(my_array); // outputs: a,b,undefined,undefined,undefined
```

NUMERIC (Array.NUMERIC property)

```
public static NUMERIC : Number
```

Represents numeric sorting instead of string-based sorting. String-based sorting, which is the default setting, treats numbers as strings when sorting them. For example, string-based sorting places 10 before 3. A numeric sort treats the elements as numbers so that 3 will be placed before 10. You can use this constant for the `options` parameter in the `sort()` or `sortOn()` method. The value of this constant is 16.

Availability
Flash Lite 2.0

See also
`sort (Array.sort method)`, `sortOn (Array.sortOn method)`

pop (Array.pop method)

```
public pop() : Object
```

Removes the last element from an array and returns the value of that element.

Availability
Flash Lite 2.0

Returns
Object - The value of the last element in the specified array.

Example

The following code creates the array `myPets_array` containing four elements, and then removes its last element:

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var popped:Object = myPets_array.pop();
trace(popped); // Displays fish.
trace(myPets_array); // Displays cat,dog,bird.
```

See also

`push (Array.push method)`, `shift (Array.shift method)`, `unshift (Array.unshift method)`

push (Array.push method)

```
public push(value:Object) : Number
```

Adds one or more elements to the end of an array and returns the new length of the array.

Availability

Flash Lite 2.0

Parameters

value : Object - One or more values to append to the array.

Returns

Number - An integer representing the length of the new array.

Example

The following example creates the array `myPets_array` with two elements, `cat` and `dog`. The second line adds two elements to the array.

Because the `push()` method returns the new length of the array, the `trace()` statement in the last line sends the new length of `myPets_array` (`4`) to the Output panel.

```
var myPets_array:Array = new Array("cat", "dog");
var pushed:Number = myPets_array.push("bird", "fish");
trace(pushed); // Displays 4.
```

See also

`pop (Array.pop method)`, `shift (Array.shift method)`, `unshift (Array.unshift method)`

RETURNINDEXEDARRAY (Array.RETURNINDEXEDARRAY property)

```
public static RETURNINDEXEDARRAY : Number
```

Represents the option to return an indexed array as a result of calling the `sort()` or `sortOn()` method. You can use this constant for the `options` parameter in the `sort()` or `sortOn()` method. This provides preview or copy functionality by returning an array that represents the results of the sort and leaves the original array unmodified. The value of this constant is 8.

Availability

Flash Lite 2.0

See also

sort (Array.sort method), sortOn (Array.sortOn method)

reverse (Array.reverse method)

```
public reverse() : Void
```

Reverses the array in place.

Availability

Flash Lite 2.0

Example

The following example uses this method to reverse the array `numbers_array`:

```
var numbers_array:Array = new Array(1, 2, 3, 4, 5, 6);
trace(numbers_array); // Displays 1,2,3,4,5,6.
numbers_array.reverse();
trace(numbers_array); // Displays 6,5,4,3,2,1.
```

shift (Array.shift method)

```
public shift() : Object
```

Removes the first element from an array and returns that element.

Availability

Flash Lite 2.0

Returns

Object - The first element in an array.

Example

The following code creates the array `myPets_array` and then removes the first element from the array and assigns it to the variable `shifted`:

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var shifted:Object = myPets_array.shift();
trace(shifted); // Displays "cat".
trace(myPets_array); // Displays dog,bird,fish.
```

See also

pop (Array.pop method), push (Array.push method), unshift (Array.unshift method)

slice (Array.slice method)

```
public slice([startIndex:Number], [endIndex:Number]) : Array
```

Returns a new array that consists of a range of elements from the original array, without modifying the original array. The returned array includes the `startIndex` element and all elements up to, but not including, the `endIndex` element.

If you don't pass any parameters, a duplicate of the original array is created.

Availability
Flash Lite 2.0

Parameters
startIndex: Number [optional] - A number specifying the index of the starting point for the slice. If *start* is a negative number, the starting point begins at the end of the array, where -1 is the last element.

endIndex: Number [optional] - A number specifying the index of the ending point for the slice. If you omit this parameter, the slice includes all elements from the starting point to the end of the array. If *end* is a negative number, the ending point is specified from the end of the array, where -1 is the last element.

Returns
Array - An array that consists of a range of elements from the original array.

Example
The following example creates an array of five pets and uses `slice()` to populate a new array that contains only four-legged pets:

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFourLeggedPets_array:Array = new Array();
var myFourLeggedPets_array = myPets_array.slice(0, 2);
trace(myFourLeggedPets_array); // Returns cat,dog.
trace(myPets_array); // Returns cat,dog,fish,canary,parrot.
```

The following example creates an array of five pets, and then uses `slice()` with a negative `start` parameter to copy the last two elements from the array:

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFlyingPets_array:Array = myPets_array.slice(-2);
trace(myFlyingPets_array); // Traces canary,parrot.
```

The following example creates an array of five pets and uses `slice()` with a negative `end` parameter to copy the middle element from the array:

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myAquaticPets_array:Array = myPets_array.slice(2,-2);
trace(myAquaticPets_array); // Returns fish.
```

sort (Array.sort method)

```
public sort([compareFunction:Object], [options:Number]) : Array
```

Sorts the elements in an array. Flash sorts according to Unicode values. (ASCII is a subset of Unicode.)

By default, `Array.sort()` works as described in the following list:

- Sorting is case-sensitive (*Z* precedes *a*).
- Sorting is ascending (*a* precedes *b*).
- The array is modified to reflect the sort order; multiple elements that have identical sort fields are placed consecutively in the sorted array in no particular order.
- Numeric fields are sorted as if they were strings, so 100 precedes 99, because "1" is a lower string value than "9".

If you want to sort an array by using settings that deviate from the default settings, you can either use one of the sorting options described in the entry for the `options` parameter or you can create your own custom function to do the sorting. If you create a custom function, you can use it by calling the `sort()` method, using the name of your custom function as the first parameter (`compareFunction`).

Availability

Flash Lite 2.0

Parameters

compareFunction : Object [optional] - A comparison function used to determine the sorting order of elements in an array. Given the elements A and B, the result of `compareFunction` can have one of the following three values:

- -1, if A should appear before B in the sorted sequence
- 0, if A equals B
- 1, if A should appear after B in the sorted sequence

options : Number [optional] - One or more numbers or names of defined constants, separated by the | (bitwise OR) operator, that change the behavior of the sort from the default. The following values are acceptable for the `options` parameter:

- `Array.CASEINSENSITIVE` or 1
- `Array.DESCENDING` or 2
- `Array.UNIQUESORT` or 4
- `Array.RETURNINDEXEDARRAY` or 8
- `Array.NUMERIC` or 16

For more information about this parameter, see the `Array.sortOn()` method.

Note: `Array.sort()` is defined in ECMA-262, but the array sorting options introduced in Flash Player 7 are Flash-specific extensions to the ECMA-262 specification.

Returns

Array - The return value depends on whether you pass any parameters, as described in the following list:

- If you specify a value of 4 or `Array.UNIQUESORT` for the `options` parameter and two or more elements being sorted have identical sort fields, Flash returns a value of 0 and does not modify the array.
- If you specify a value of 8 or `Array.RETURNINDEXEDARRAY` for the `options` parameter, Flash returns an array that reflects the results of the sort and does not modify the array.
- Otherwise, Flash returns nothing and modifies the array to reflect the sort order.

Example

Usage 1: The following example shows the use of `Array.sort()` with and without a value passed for `options`:

```
var fruits_array:Array = new Array("oranges", "apples", "strawberries", "pineapples",
"cherries");
trace(fruits_array); // Displays oranges,apples,strawberries,pineapples,cherries.
fruits_array.sort();
trace(fruits_array); // Displays apples,cherries,oranges,pineapples,strawberries.
fruits_array.sort(Array.DESCENDING);
trace(fruits_array); // Displays strawberries,pineapples,oranges,cherries,apples.
```

Usage 2: The following example uses `Array.sort()` with a compare function. The entries are sorted in the form name:password. Sort using only the name part of the entry as a key:

```
var passwords_array:Array = new Array("mom:glam", "ana:ring", "jay:mag", "anne:home",
"regina:silly");
function order(a, b):Number {
    var name1:String = a.split(":")[0];
    var name2:String = b.split(":")[0];
    if (name1<name2) {
    return -1;
    } else if (name1>name2) {
    return 1;
    } else {
    return 0;
    }
}
trace("Unsorted:");
trace(passwords_array);
//Displays mom:glam,ana:ring,jay:mag,anne:home,regina:silly.
passwords_array.sort(order);
trace("Sorted:");
trace(passwords_array);
//Displays ana:ring,anne:home,jay:mag,mom:glam,regina:silly.
```

See also

`| bitwise OR operator`, `sortOn (Array.sortOn method)`

sortOn (Array.sortOn method)

```
public sortOn(fieldName:Object, [options:Object]) : Array
```

Sorts the elements in an array according to one or more fields in the array. The array should have the following characteristics:

- The array is an indexed array, not an associative array.
- Each element of the array holds an object with one or more properties.
- All of the objects have at least one property in common, the values of which can be used to sort the array. Such a property is called a *field*.

If you pass multiple `fieldName` parameters, the first field represents the primary sort field, the second represents the next sort field, and so on. Flash sorts according to Unicode values. (ASCII is a subset of Unicode.) If either of the elements being compared does not contain the field specified in the `fieldName` parameter, the field is assumed to be `undefined`, and the elements are placed consecutively in the sorted array in no particular order.

By default, `Array.sortOn()` works as described in the following list:

- Sorting is case-sensitive (*Z* precedes *a*).
- Sorting is ascending (*a* precedes *b*).
- The array is modified to reflect the sort order; multiple elements that have identical sort fields are placed consecutively in the sorted array in no particular order.
- Numeric fields are sorted as if they were strings, so 100 precedes 99, because "1" is a lower string value than "9".

You can use the `options` parameter to override the default sort behavior. If you want to sort a simple array (for example, an array with only one field), or if you want to specify a sort order that the `options` parameter doesn't support, use `Array.sort()`.

To pass multiple flags, separate them with the bitwise OR (`|`) operator:

```
my_array.sortOn(someFieldName, Array.DESCENDING | Array.NUMERIC);
```

Availability
Flash Lite 2.0

Parameters
fieldName: Object - A string that identifies a field to be used as the sort value, or an array in which the first element represents the primary sort field, the second represents the secondary sort field, and so on.

options: Object [optional] - One or more numbers or names of defined constants, separated by the `|` (bitwise OR) operator, that change the sorting behavior. The following values are acceptable for the `options` parameter:

- `Array.CASEINSENSITIVE` or 1
- `Array.DESCENDING` or 2
- `Array.UNIQUESORT` or 4
- `Array.RETURNINDEXEDARRAY` or 8
- `Array.NUMERIC` or 16

Code hinting is enabled if you use the string form of the flag (for example, `DESCENDING`) rather than the numeric form (2).

Returns
Array - The return value depends on whether you pass any parameters, as described in the following list:

- If you specify a value of 4 or `Array.UNIQUESORT` for the `options` parameter, and two or more elements being sorted have identical sort fields, Flash returns a value of 0 and does not modify the array.
- If you specify a value of 8 or `Array.RETURNINDEXEDARRAY` for the `options` parameter, Flash returns an array that reflects the results of the sort and does not modify the array.
- Otherwise, Flash returns nothing and modifies the array to reflect the sort order.

Example
The following example creates a new array and sorts it according to the `name` and `city` fields. The first sort uses `name` as the first sort value and `city` as the second. The second sort uses `city` as the first sort value and `name` as the second.

```
var rec_array:Array = new Array();
rec_array.push({name: "john", city: "omaha", zip: 68144});
rec_array.push({name: "john", city: "kansas city", zip: 72345});
rec_array.push({name: "bob", city: "omaha", zip: 94010});
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// john, omaha
// john, kansas city
// bob, omaha

rec_array.sortOn(["name", "city"]);
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// bob, omaha
// john, kansas city
// john, omaha

rec_array.sortOn(["city", "name" ]);
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// john, kansas city
// bob, omaha
// john, omaha
```

The following array of objects is used by subsequent examples that show how to use the `options` parameter:

```
var my_array:Array = new Array();
my_array.push({password: "Bob", age:29});
my_array.push({password: "abcd", age:3});
my_array.push({password: "barb", age:35});
my_array.push({password: "catchy", age:4});
```

Performing a default sort on the password field produces the following results:

```
my_array.sortOn("password");
// Bob
// abcd
// barb
// catchy
```

Performing a case-insensitive sort on the password field produces the following results:

```
my_array.sortOn("password", Array.CASEINSENSITIVE);
// abcd
// barb
// Bob
// catchy
```

Performing a case-insensitive, descending sort on the password field produces the following results:

```
my_array.sortOn("password", Array.CASEINSENSITIVE | Array.DESCENDING);
// catchy
// Bob
// barb
// abcd
```

Performing a default sort on the age field produces the following results:

```
my_array.sortOn("age");
// 29
// 3
// 35
// 4
```

Performing a numeric sort on the age field produces the following results:

```
my_array.sortOn("age", Array.NUMERIC);
// my_array[0].age = 3
// my_array[1].age = 4
// my_array[2].age = 29
// my_array[3].age = 35
```

Performing a descending numeric sort on the age field produces the following results:

```
my_array.sortOn("age", Array.DESCENDING | Array.NUMERIC);
// my_array[0].age = 35
// my_array[1].age = 29
// my_array[2].age = 4
// my_array[3].age = 3
```

When using the `Array.RETURNINDEXARRAY` sorting option, you must assign the return value to a different array. The original array is not modified.

```
var indexArray:Array = my_array.sortOn("age", Array.RETURNINDEXEDARRAY);
```

See also

`| bitwise OR operator`, `sort (Array.sort method)`

splice (Array.splice method)

```
public splice(startIndex:Number, [deleteCount:Number], [value:Object]) : Array
```

Adds elements to and removes elements from an array. This method modifies the array without making a copy.

Availability

Flash Lite 2.0

Parameters

startIndex : Number - An integer that specifies the index of the element in the array where the insertion or deletion begins. You can specify a negative integer to specify a position relative to the end of the array (for example, -1 is the last element of the array).

deleteCount : Number [optional] - An integer that specifies the number of elements to be deleted. This number includes the element specified in the `startIndex` parameter. If no value is specified for the `deleteCount` parameter, the method deletes all of the values from the `startIndex` element to the last element in the array. If the value is 0, no elements are deleted.

value : Object [optional] - Specifies the values to insert into the array at the insertion point specified in the `startIndex` parameter.

Returns

Array - An array containing the elements that were removed from the original array.

Example

The following example creates an array and splices it by using element index 1 for the `startIndex` parameter. This removes all elements from the array starting with the second element, leaving only the element at index 0 in the original array:

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
trace( myPets_array.splice(1) ); // Displays dog,bird,fish.
trace( myPets_array ); // cat
```

The following example creates an array and splices it by using element index 1 for the `startIndex` parameter and the number 2 for the `deleteCount` parameter. This removes two elements from the array, starting with the second element, leaving the first and last elements in the original array:

```
var myFlowers_array:Array = new Array("roses", "tulips", "lilies", "orchids");
trace( myFlowers_array.splice(1,2 ) ); // Displays tulips,lilies.
trace( myFlowers_array ); // roses,orchids
```

The following example creates an array and splices it by using element index 1 for the `startIndex` parameter, the number 0 for the `deleteCount` parameter, and the string `chair` for the `value` parameter. This does not remove anything from the original array, and adds the string `chair` at index 1:

```
var myFurniture_array:Array = new Array("couch", "bed", "desk", "lamp");
trace( myFurniture_array.splice(1,0, "chair" ) ); // Displays empty array.
trace( myFurniture_array ); // displays couch,chair,bed,desk,lamp
```

toString (Array.toString method)

```
public toString() : String
```

Returns a string value representing the elements in the specified Array object. Every element in the array, starting with index 0 and ending with the highest index, is converted to a concatenated string and separated by commas. To specify a custom separator, use the `Array.join()` method.

Availability

Flash Lite 2.0

Returns

String - A string.

Example

The following example creates `my_array` and converts it to a string.

```
var my_array:Array = new Array();
my_array[0] = 1;
my_array[1] = 2;
my_array[2] = 3;
my_array[3] = 4;
my_array[4] = 5;
trace(my_array.toString()); // Displays 1,2,3,4,5.
```

This example outputs 1,2,3,4,5 as a result of the `trace` statement.

See also

`split (String.split method)`, `join (Array.join method)`

UNIQUESORT (Array.UNIQUESORT property)

`public static UNIQUESORT : Number`

Represents the unique sorting requirement. You can use this constant for the options parameter in the `sort()` or `sortOn()` method. The unique sorting option aborts the sort if any two elements or fields being sorted have identical values. The value of this constant is 4.

Availability

Flash Lite 2.0

See also

`sort (Array.sort method)`, `sortOn (Array.sortOn method)`

unshift (Array.unshift method)

`public unshift(value:Object) : Number`

Adds one or more elements to the beginning of an array and returns the new length of the array.

Availability

Flash Lite 2.0

Parameters

value : Object - One or more numbers, elements, or variables to be inserted at the beginning of the array.

Returns

Number - An integer representing the new length of the array.

Example

The following example shows the use of the `Array.unshift()` method:

```
var pets_array:Array = new Array("dog", "cat", "fish");
trace( pets_array ); // Displays dog,cat,fish.
pets_array.unshift("ferrets", "gophers", "engineers");
trace( pets_array ); // Displays ferrets,gophers,engineers,dog,cat,fish.
```

See also

`pop (Array.pop method)`, `push (Array.push method)`, `shift (Array.shift method)`

BitmapData (flash.display.BitmapData)

```
Object
    |
    +-flash.display.BitmapData
```

```
public class BitmapData
extends Object
```

The BitmapData class lets you create arbitrarily sized transparent or opaque bitmap images and manipulate them in various ways at runtime.

This class lets you separate bitmap rendering operations from the Flash Lite player internal display updating routines. By manipulating a BitmapData object directly, you can create very complex images without incurring the per-frame overhead of constantly redrawing the content from vector data.

The methods of the BitmapData class support a variety of effects that are not available through the generic filter interface.

A BitmapData object contains an array of pixel data. This data can represent either a fully opaque bitmap or a transparent bitmap that contains alpha channel data. Either type of BitmapData object is stored as a buffer of 32-bit integers. Each 32-bit integer determines the properties of a single pixel in the bitmap.

Each 32-bit integer is a combination of four 8-bit channel values (from 0 to 255) that describe the alpha transparency and the red, green, and blue (ARGB) values of the pixel.

The four channels (red, green, blue, and alpha) are represented as numbers when you use them with the `BitmapData.copyChannel()` method or the `DisplacementMapFilter.componentX` and `DisplacementMapFilter.componentY` properties, as follows:

- `1` (red)
- `2` (green)
- `4` (blue)
- `8` (alpha)

You can attach BitmapData objects to a MovieClip object by using the `MovieClip.attachBitmap()` method.

You can use a BitmapData object to fill an area in a movie clip by using the `MovieClip.beginBitmapFill()` method.

The maximum width and maximum height of a BitmapData object is 2880 pixels.

Availability

Flash Lite 3.1

See also

attachBitmap (MovieClip.attachBitmap method), beginFill (MovieClip.beginFill method)

Property summary

Modifiers	Property	Description
	height : Number [read-only]	The height of the bitmap image in pixels.
	rectangle : Rectangle [read-only]	The rectangle that defines the size and location of the bitmap image.
	transparent : Boolean [read-only]	Defines whether the bitmap image supports per-pixel transparency.
	width : Boolean [read-only]	The width of the bitmap image in pixels.

`constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)`

Constructor summary

Signature	Description
BitmapData(`width:Number, height:Number, [transparent:Boolean], [fillColor:Number]`)	Creates a BitmapData object with a specified width and height.

Method summary

Modifiers	Signature	Description
	`applyFilter`	Takes a source image and a filter object and generates the filtered image. Flash Lite 3.1 does not support filters, so this method is not supported.
	clone(): BitmapData	Returns a new BitmapData object that is a clone of the original instance with an exact copy of the contained bitmap.
	colorTransform(`rect:Rectangle, colorTransform:ColorTransform`) : Void	Adjusts the color values in a specified area of a bitmap image by using a ColorTransform object.
	copyChannel(`sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, sourceChannel:Number, destChannel:Number`): Void	Transfers data from one channel of another BitmapData object or the current BitmapData object into a channel of the current BitmapData object.

Modifiers	Signature	Description
	copyPixels(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, [alphaBitmap:BitmapData] , [alphaPoint:Point, [mergeAlpha:Boolean]) : Void	Provides a fast routine to perform pixel manipulation between images with no stretching, rotation, or color effects.
	dispose() : Void	Frees memory that is used to store the BitmapData object.
	draw(source:Object, [matrix:Matrix], [colorTransform:ColorTransform], [blendMode:Object], [cliprect:Rectangle], [smooth:Boolean]) : Void	Draws a source image or movie clip onto a destination image, using the Flash Lite player vector renderer.
	fillRect(rect:Rectangle, color:Number) : Void	Fills a rectangular area of pixels with a specified ARGB color.
	floodFill(x:Number, y:Number, color:Number) : Void	Performs a flood fill operation on an image starting at an (*x*, *y*) coordinate and filling with a certain color.
	generateFilterRect	Determines the destination rectangle that the applyFilter() method call affects, given a BitmapData object, a source rectangle, and a filter object. Flash Lite does not support this method.
	getColorBoundsRect(mask:Number, color:Number, [findColor:Boolean]) :Rectangle	Determines a rectangular region that fully encloses all pixels of a specified color within the bitmap image.
	getPixel(x:Number, y:Number) : Number	Returns an integer that represents an RGB pixel value from a BitmapData object at a specific point (*x*, *y*).
	getPixel32(x:Number, y:Number) : Number	Returns an ARGB color value that contains alpha channel data and RGB data.
	hitTest(firstPoint:Point, firstAlphaThreshold:Number, secondObject:Object, [secondBitmapPoint:Point] , [secondAlphaThreshold:Number]) :Boolean	Performs pixel-level hit detection between one bitmap image and a point, rectangle or other bitmap image.
static	loadBitmap(id:String): BitmapData	Returns a new BitmapData object that contains a bitmap image representation of the symbol that is identified by a specified linkage ID in the library.

Modifiers	Signature	Description
	merge(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, redMult:Number, greenMult:Number, blueMult:Number, alphaMult:Number) : Void	Performs per-channel blending from a source image to a destination image.
	noise	Fills an image with pixels representing random noise. Flash Lite does not support this method.
	paletteMap	Remaps the color channel values in an image that has up to four arrays of color palette data, one for each channel. Flash Lite does not support this method.
	perlinNoise	Generates a Perlin noise image. Flash Lite does not support this method.
	pixelDissolve	Performs a pixel dissolve either from a source image to a destination image or by using the same image. Flash Lite does not support this method.
	scroll	Scrolls an image by a certain (*x*, *y*) pixel amount. Flash Lite does not support this method.
	setPixel(x:Number, y:Number, color:Number) : Void	Sets the color of a single pixel of a BitmapData object.
	setPixel32(x:Number, y:Number, color:Number) : Void	Sets the color and alpha transparency values of a single pixel of a BitmapData object.
	threshold	Tests pixel values in an image against a specified threshold and sets pixels that pass the test to new color values. Flash Lite does not support this method.

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

BitmapData constructor

`public BitmapData(width:Number, height:Number, [transparent:Boolean], [fillColor:Number])`

Creates a BitmapData object with a specified width and height. If you specify a value for the `fillColor` parameter, every pixel in the bitmap is set to that color.

By default, the bitmap is created as opaque, unless you pass the value `true` for the `transparent` parameter. Once you create an opaque bitmap, you cannot change it to a transparent bitmap. Every pixel in an opaque bitmap uses only 24 bits of color channel information. If you define the bitmap as `transparent`, every pixel uses 32 bits of color channel information, including an alpha transparency channel.

The maximum width and maximum height of a BitmapData object is 2880 pixels. If you specify a width or height value that is greater than 2880, a new instance is not created.

Availability
Flash Lite 3.1

Parameters
width : Number - The width of the bitmap image in pixels.

height : Number - The height of the bitmap image in pixels.

transparent : Boolean [optional] - Specifies whether the bitmap image supports per-pixel transparency. The default value is `true` (transparent). To create a fully transparent bitmap, set the value of the `transparent` parameter to `true` and the value of the `fillColor` parameter to `0x00000000` (or to 0).

fillColor : Number [optional] - A 32-bit ARGB color value that you use to fill the bitmap image area. The default value is 0xFFFFFFFF (solid white).

Example
The following example creates a new BitmapData object. The values in this example are the default values for the `transparent` and `fillColor` parameters; you could call the constructor without these parameters and get the same result.

```
import flash.display.BitmapData;

var width:Number = 100;
var height:Number = 80;
var transparent:Boolean = true;
var fillColor:Number = 0xFFFFFFFF;

var bitmap_1:BitmapData = new BitmapData(width, height, transparent, fillColor);

trace(bitmap_1.width); // 100
trace(bitmap_1.height); // 80
trace(bitmap_1.transparent); // true

var bitmap_2:BitmapData = new BitmapData(width, height);

trace(bitmap_2.width); // 100
trace(bitmap_2.height); // 80
trace(bitmap_2.transparent); // true
```

clone (BitmapData.clone method)

```
public clone() : BitmapData
```

Returns a new BitmapData object that is a copy of the cloned bitmap. A clone and and the object cloned have identical properties. However, a clone does not evaluate as equal to the BitmapData object that was cloned because the properties of the original object are passed by value to the clone, they are not passed by reference. If you change the values in the original object after the clone is created, the clone does not receive the new values.

Availability
Flash Lite 3.1

Returns

BitmapData - A new BitmapData object that is identical to the original.

Example

The following example creates three BitmapData objects and compares them. The code uses the `BitmapData constructor to create the bitmap_1` instance. It creates the `bitmap_2` instance by setting it equal to `bitmap_1`. It creates the `clonedBitmap` instance by cloning `bitmap_1`. Notice that although `bitmap_2` evaluates as being equal to `bitmap_1`, `clonedBitmap` does not, even though it contains the same values as `bitmap_1`.

```
import flash.display.BitmapData;

var bitmap_1:BitmapData = new BitmapData(100, 80, false, 0x000000);
var bitmap_2:BitmapData = bitmap_1;
var clonedBitmap:BitmapData = bitmap_1.clone();

trace("bitmap_1 == bitmap_2 " + (bitmap_1 == bitmap_2)); // true
trace("bitmap_1 == clonedBitmap " + (bitmap_1 == clonedBitmap)); // false

trace("-------------bitmap_1 properties-------------")
for(var i in bitmap_1) {
    trace(">> " + i + ": " + bitmap_1[i]);
}

trace("-------------bitmap_2 properties-------------")
for(var i in bitmap_2) {
    trace(">> " + i + ": " + bitmap_1[i]);
}

trace("-------------clonedBitmap properties-------------")
for(var i in clonedBitmap) {
    trace(">> " + i + ": " + clonedBitmap[i]);
}
```

To further demonstrate the relationships between `bitmap_1`, `bitmap_2`, and `clonedBitmap`, the following example modifies the pixel value at (1, 1) of `bitmap_1`. Modifying pixel value at (1, 1) changes the pixel value for `bitmap_2`, because `bitmap_2` contains references to `bitmap_1`. Modifying `bitmap_1` does not change clonedBitmap because `clonedBitmap` does not reference the values in `bitmap_1`.

```
import flash.display.BitmapData;

var bitmap_1:BitmapData = new BitmapData(100, 80, false, 0x000000);
var bitmap_2:BitmapData = bitmap_1;
var clonedBitmap:BitmapData = bitmap_1.clone();

trace(bitmap_1.getPixel32(1, 1)); // -16777216
trace(bitmap_2.getPixel32(1, 1)); // -16777216
trace(clonedBitmap.getPixel32(1, 1)); // -16777216

bitmap_1.setPixel32(1, 1, 0xFFFFFF);

trace(bitmap_1.getPixel32(1, 1)); // -1
trace(bitmap_2.getPixel32(1, 1)); // -1
trace(clonedBitmap.getPixel32(1, 1)); // -16777216
```

colorTransform (BitmapData.colorTransform method)

```
public colorTransform(rect: Rectangle, colorTransform: ColorTransform) : Void
```

Adjusts the color values in a specified area of a bitmap image by using a ColorTransform object. If the rectangle matches the boundaries of the bitmap image, this method transforms the color values of the entire image.

Availability

Flash Lite 3.1

Parameters

rect : Rectangle - A Rectangle object that defines the area of the image in which the ColorTransform object is applied.

colorTransform : ColorTransform - A ColorTransform object that describes the color transformation values to apply.

Example

The following example shows how to apply a color transform operation to a BitmapData instance.

```
fscommand2("SetSoftKeys");
import flash.display.BitmapData;
import flash.geom.ColorTransform;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        myBitmapData.colorTransform(myBitmapData.rectangle, new ColorTransform(1, 0, 0, 1, 255, 0,
0, 0));
    }
};
```

See also

ColorTransform (flash.geom.ColorTransform), Rectangle (flash.geom.Rectangle)

copyChannel (BitmapData.copyChannel method)

```
public copyChannel(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point,
sourceChannel:Number, destChannel:Number) : Void
```

Transfers data from one channel of another BitmapData object or the current BitmapData object into a channel of the current BitmapData object. All of the data in the other channels in the destination BitmapData object are preserved.

The source channel value and destination channel value can be one of following values or a sum of any of the values:

- 1 (red)
- 2 (green)
- 4 (blue)
- 8 (alpha)

Availability
Flash Lite 3.1

Parameters
sourceBitmap : BitmapData - The input bitmap image to use. The source image can be a different BitmapData object, or it can refer to the current BitmapData object.

sourceRect : Rectangle - The source Rectangle object. If you only want to copy channel data from a smaller area within the bitmap, specify a source rectangle that is smaller than the overall size of the BitmapData object.

destPoint : Point - The destination Point object that represents the upper-left corner of the rectangular area where the new channel data is placed. If you want to copy channel data from one area to a different area in the destination image, specify a point other than (0,0).

sourceChannel : Number - The source channel. Use a value from the set (1,2,4,8), which represent red, green, blue, and alpha channels, respectively, or a sum of any of the values.

destChannel : Number - The destination channel. Use a value from the set (1,2,4,8), which represent red, green, blue, and alpha channels, respectively, or a sum of any of the values.

Example
The following example shows how to copy a source ARGB channel from a BitmapData object back onto itself at a different location:

```
fscommand2("SetSoftKeys");
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        myBitmapData.copyChannel(myBitmapData, new Rectangle(0, 0, 50, 80), new Point(51, 0),
3, 1);
    }
};
```

See also
Rectangle (flash.geom.Rectangle)

copyPixels (BitmapData.copyPixels method)

```
public copyPixels(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point) : Void
```

Provides a fast routine to perform pixel manipulation between images with no stretching, rotation, or color effects. This method copies a rectangular area of a source image to a rectangular area of the same size at the destination point of the destination BitmapData object.

Availability
Flash Lite 3.1

Parameters
sourceBitmap : BitmapData - The input bitmap image from which to copy pixels. The source image can be a different BitmapData instance, or it can refer to the current BitmapData instance.

sourceRect : Rectangle - A rectangle that defines the area of the source image to use as input.

destPoint : Point - The destination point, that represents the upper-left corner of the rectangular area where the new pixels are placed.

Example
The following example shows how to copy pixels from one BitmapData instance to another.

```
fscommand2("SetSoftKeys");

import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var bitmapData_1:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);
var bitmapData_2:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc_1:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_1.attachBitmap(bitmapData_1, this.getNextHighestDepth());

var mc_2:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_2.attachBitmap(bitmapData_2, this.getNextHighestDepth());
mc_2._x = 101;

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        bitmapData_2.copyPixels(bitmapData_1, new Rectangle(0, 0, 50, 80), new Point(51, 0));
    }
    else if (keyCode == ExtendedKey.SOFT2) {
        // Handle right soft key event
        bitmapData_1.copyPixels(bitmapData_2, new Rectangle(0, 0, 50, 80), new Point(51, 0));
    }
};
```

dispose (BitmapData.dispose method)

```
public dispose() : Void
```

Frees memory that is used to store the BitmapData object.

Call `myBitmapData.dispose()` to set the width and height of the image to 0. After a BitmapData object's memory has been freed, method and property access calls on the instance fail, returning a value of -1.

Availability
Flash Lite 3.1

Example

The following example shows how to release the memory of a BitmapData instance, which results in a cleared instance.

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        myBitmapData.dispose()
        trace(myBitmapData.width); // -1
        trace(myBitmapData.height); // -1
        trace(myBitmapData.transparent); // 1
    }
};
```

draw (BitmapData.draw method)

```
public draw(source:Object, [matrix:Matrix], [colorTransform:ColorTransform],
[clipRect:Rectangle], [smooth:Boolean]) : Void
```

Draws a source image or movie clip onto a destination image, using the Flash Lite player vector renderer. You can use Matrix, ColorTransform, BlendMode objects, and a destination Rectangle object to control how the rendering performs. Optionally, you can specify whether the bitmap should be smoothed when scaled. This works only if the source object is a BitmapData object.

This method directly corresponds to how objects are drawn using the standard vector renderer for objects in the authoring tool interface.

A source MovieClip object does not use any of its on-stage transformations for this call. It is treated as it exists in the library or file, with no matrix transform, no color transform, and no blend mode. If you want to draw the movie clip by using its own transform properties, you can use its Transform object to pass the various transformation properties.

The `blendMode` parameter is not supported in Flash Lite.

Availability

Flash Lite 3.1

Parameters

source : Object - The BitmapData object to draw.

matrix : Matrix [optional] - A Matrix object used to scale, rotate, or translate the coordinates of the bitmap. If no object is supplied, the bitmap image will not be transformed. Set this parameter to an identity matrix, created using the default `new Matrix()` constructor, if you must pass this parameter but you do not want to transform the image.

colorTransform : ColorTransform [optional] - A ColorTransform object that you use to adjust the color values of the bitmap. If no object is supplied, the bitmap image's colors will not be transformed. Set this parameter to a ColorTransform object created using the default `new ColorTransform()` constructor, if you must pass this parameter but you do not want to transform the image.

clipRect : Rectangle [optional] - A Rectangle object. If you do not supply this value, no clipping occurs.

smooth : Boolean [optional] - A Boolean value that determines whether a BitmapData object is smoothed when scaled. The default value is `false`.

Example

The following example shows how to draw from a source MovieClip instance to a BitmapData object.

```
fscommand2("SetSoftKeys");

import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Matrix;
import flash.geom.ColorTransform;
var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);
var mc_1: MovieClip = this.createEmptyMovieClip("mc",this.getNextHighestDepth());
mc_1.attachBitmap (my BitmapData,this.getNextHighestDepth());

var mc_2:MovieClip = createRectangle(50, 40, 0xFF0000);
mc_2._x = 101;

var myMatrix:Matrix = new Matrix();
myMatrix.rotate(Math.PI/2);
var translateMatrix:Matrix = new Matrix();
translateMatrix.translate(70, 15);
myMatrix.concat(translateMatrix);

var myColorTransform:ColorTransform = new ColorTransform (0, 0, 1 , 1, 0,  0 , 255, 0);
var blend Mode:String = "normal";
var myRectangle:Rectangle = new Rectangle(0, 0, 100, 80);
var smooth:Boolean = true;
mc_1.onPress = function() {
    myBitmapData.draw(mc_2, myMatrix, myColorTransform, blendMode, myRectangle,
    smooth);
}

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

fillRect (BitmapData.fillRect method)

```
public fillRect(rect:Rectangle, color:Number) : Void
```

Fills a rectangular area of pixels with a specified ARGB color.

Availability

Flash Lite 3.1

Parameters

rect : Rectangle - The rectangular area to fill.

color : Number - The ARGB color value that fills the area. ARGB colors are often specified in hexadecimal format; for example, 0xFF336699.

Example

The following example shows how to fill an area that is defined by a `Rectangle` within a `BitmapData` with a color.

```
fscommand2("SetSoftKeys");

import flash.display.BitmapData;
import flash.geom.Rectangle;
var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);
var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        myBitmapData.fillRect(new Rectangle(0, 0, 50, 40), 0x00FF0000);
    }
};
```

See also

`Rectangle (flash.geom.Rectangle)`

floodFill (BitmapData.floodFill method)

```
public floodFill(x:Number, y:Number, color:Number) : Void
```

Performs a flood fill operation on an image starting at an (*x*, *y*) coordinate and filling with a certain color. The `floodFill()` method is similar to the paint bucket tool in various painting programs. The color is an ARGB color that contains alpha information and color information.

Availability

Flash Lite 3.1

Parameters

x : Number - The *x* coordinate of the image.

y : Number - The *y* coordinate of the image.

color : Number - The ARGB color to use as a fill. ARGB colors are often specified in hexadecimal format, like 0xFF336699.

Example

The following example shows how to apply a flood fill a color into to an image starting at the point where a user clicks the mouse within a BitmapData object.

```
fscommand2("SetSoftKeys");

import flash.display.BitmapData;
import flash.geom.Rectangle;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);
var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

myBitmapData.fillRect(new Rectangle(0, 0, 50, 40), 0x00FF0000);

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        myBitmapData.floodFill(_xmouse, _ymouse, 0x000000FF);}
};
```

getColorBoundsRect (BitmapData.getColorBoundsRect method)

```
public getColorBoundsRect(mask:Number, color:Number, [findColor:Boolean]) : Rectangle
```

Determines a rectangular region that fully encloses all pixels of a specified color within the bitmap image.

For example, if you have a source image and you want to determine the rectangle of the image that contains a nonzero alpha channel, you pass {`mask: 0xFF000000, color: 0x00000000`} as parameters. The entire image is searched for the bounds of pixels whose `(value & mask) != color`. To determine white space around an image, you pass {`mask: 0xFFFFFFFF, color: 0xFFFFFFFF`} to find the bounds of nonwhite pixels.

Availability

Flash Lite 3.1

Parameters

mask : Number - A hexadecimal color value.

color : Number - A hexadecimal color value.

findColor : Boolean [optional] - If the value is set to `true`, returns the bounds of a color value in an image. If the value is set to `false`, returns the bounds of where this color doesn't exist in an image. The default value is `true`.

Returns

Rectangle - The region of the image that is the specified color.

Example

The following example shows how to determine a rectangular region that fully encloses all pixels of a specified color within the bitmap image:

```
fscommand2("SetSoftKeys");

import flash.display.BitmapData;
import flash.geom.Rectangle;
var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);
var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.fillRect(new Rectangle(0, 0, 50, 40), 0x00FF0000);

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        var colorBoundsRect:Rectangle = myBitmapData.getColorBoundsRect(0x00FFFFFF,
0x00FF0000, true);
trace(colorBoundsRect); // (x=0, y=0, w=50, h=40)
    }
};
Key.addListener (myListener);
```

getPixel (BitmapData.getPixel method)

```
public getPixel(x:Number, y:Number) : Number
```

Returns an integer that represents an RGB pixel value from a BitmapData object at a specific point (*x*, *y*). The `getPixel()` method returns an unmultiplied pixel value. No alpha information is returned.

All pixels in a BitmapData object are stored as premultiplied color values. A premultiplied image pixel has the red, green, and blue color channel values already multiplied by the alpha data. For example, if the alpha value is 0, the values for the RGB channels are also 0, independent of their unmultiplied values.

This loss of data can cause some problems when you are performing operations. All Flash Lite player methods take and return unmultiplied values. The internal pixel representation is unmultiplied before it is returned as a value. During a set operation, the pixel value is premultiplied before setting the raw image pixel.

Availability

Flash Lite 3.1

Parameters

x : Number - The *x* position of the pixel.

y : Number - The *y* position of the pixel.

Returns

Number - A number that represents an RGB pixel value. If the (*x*, *y*) coordinates are outside the bounds of the image, 0 is returned.

Example

The following example uses the `getPixel()` method to retrieve the RGB value of a pixel at a specific *x* and *y* position.

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace("0x" + myBitmapData.getPixel(0, 0).toString(16)); // 0xcccccc
```

See also

getPixel32 (BitmapData.getPixel32 method)

getPixel32 (BitmapData.getPixel32 method)

```
public getPixel32(x:Number, y:Number) : Number
```

Returns an ARGB color value that contains alpha channel data and RGB data. This method is similar to the `getPixel()` method, which returns an RGB color without alpha channel data.

Availability

Flash Lite 3.1

Parameters

x : Number - The *x* position of the pixel.

y : Number - The *y* position of the pixel.

Returns

Number - A number that represent an ARGB pixel value. If the (*x*, *y*) coordinates are outside the bounds of the image, 0 is returned. If the bitmap was created as an opaque bitmap and not a transparent one, then this method will return an error code of -1.

Example

The following example uses the `getPixel32()` method to retrieve the ARGB value of a pixel at a specific *x* and *y* position:

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xFFAACCEE);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

var alpha:String = (myBitmapData.getPixel32(0, 0) >> 24 & 0xFF).toString(16);
trace(">> alpha: " + alpha); // ff

var red:String = (myBitmapData.getPixel32(0, 0) >> 16 & 0xFF).toString(16);
trace(">> red: " + red); // aa

var green:String = (myBitmapData.getPixel32(0, 0) >> 8 & 0xFF).toString(16);
trace(">> green: " + green); // cc

var blue:String = (myBitmapData.getPixel32(0, 0) & 0xFF).toString(16);
trace(">> blue: " + blue); // ee

trace("0x" + alpha + red + green + blue); // 0xffaaccee
```

See also

getPixel (BitmapData.getPixel method)

height (BitmapData.height property)

```
public height : Number [read-only]
```

The height of the bitmap image in pixels.

Availability

Flash Lite 3.1

Example

The following example shows that the `height` property of the `BitmapData` instance is read-only by trying to set it and failing:

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace(myBitmapData.height); // 80

myBitmapData.height = 999;
trace(myBitmapData.height); // 80
```

hitTest (BitmapData.hitTest method)

```
public hitTest(firstPoint:Point, firstAlphaThreshold:Number, secondObject:Object,
[secondBitmapPoint:Point], [secondAlphaThreshold:Number]) : Boolean
```

Performs pixel-level hit detection between one bitmap image and a point, rectangle, or other bitmap image. No stretching, rotation, or other transformation of either object is considered when doing the hit test.

If an image is an opaque image, it is considered a fully opaque rectangle for this method. Both images must be transparent images to perform pixel-level hit testing that considers transparency. When you are testing two transparent images, the alpha threshold parameters control what alpha channel values, from 0 to 255, are considered opaque.

Availability

Flash Lite 3.1

Parameters

firstPoint : Point - A point that defines a pixel location in the current BitmapData instance.

firstAlphaThreshold : Number - The highest alpha channel value that is considered opaque for this hit test.

secondObject : Object - A Rectangle, Point, or BitmapData object.

secondBitmapPoint : Point [optional] - A point that defines a pixel location in the second BitmapData object. Use this parameter only when the value of `secondObject` is a BitmapData object.

secondAlphaThreshold : Number [optional] - The highest alpha channel value that is considered opaque in the second BitmapData object. Use this parameter only when the value of `secondObject` is a BitmapData object and both BitmapData objects are transparent.

Returns

Boolean - A Boolean value. If there is a hit, returns a value of `true`; otherwise, `false`.

Example

The following example shows how to determine if a BitmapData object is colliding with a `MovieClip`.

```
import flash.display.BitmapData;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc_1:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_1.attachBitmap(myBitmapData, this.getNextHighestDepth());

var mc_2:MovieClip = createRectangle(20, 20, 0xFF0000);

var destPoint:Point = new Point(myBitmapData.rectangle.x, myBitmapData.rectangle.y);
var currPoint:Point = new Point();

mc_1.onEnterFrame = function() {
    currPoint.x = mc_2._x;
    currPoint.y = mc_2._y;
    if(myBitmapData.hitTest(destPoint, 255, currPoint)) {
        trace(">> Collision at x:" + currPoint.x + " and y:" + currPoint.y);
    }
}

mc_2.startDrag(true);

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

loadBitmap (BitmapData.loadBitmap method)

```
public static loadBitmap(id:String) : BitmapData
```

Returns a new BitmapData object that contains a bitmap image representation of the symbol that is identified by a specified linkage ID in the library.

Availability

Flash Lite 3.1

Parameters

id : String - A linkage ID of a symbol in the library.

Returns

BitmapData - A bitmap image representation of the symbol.

Example

The following example loads a bitmap with the linkageId `libraryBitmap` from your library. You must attach it to a MovieClip object to give it a visual representation.

```
import flash.display.BitmapData;

var linkageId:String = "libraryBitmap";
var myBitmapData:BitmapData = BitmapData.loadBitmap(linkageId);
trace(myBitmapData instanceof BitmapData); // true

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
```

merge (BitmapData.merge method)

```
public merge(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Rectangle,
redMult:Number, greenMult:Number, blueMult:Number, alphaMult:Number) : Void
```

Performs per-channel blending from a source image to a destination image. The following formula is used for each channel:

```
new red dest = (red source * redMult) + (red dest * (256 - redMult) / 256;
```

The `redMult`, `greenMult`, `blueMult`, and `alphaMult` values are the multipliers used for each color channel. Their valid range is from 0 to 256.

Availability

Flash Lite 3.1

Parameters

sourceBitmap : BitmapData - The input bitmap image to use. The source image can be a different BitmapData object, or it can refer to the current BitmapData object.

sourceRect : Rectangle - A rectangle that defines the area of the source image to use as input.

destPoint : Point - The point within the destination image (the current BitmapData instance) that corresponds to the upper-left corner of the source rectangle.

redMult : Number - A number by which to multiply the red channel value.

greenMult : Number - A number by which to multiply the green channel value.

blueMult : Number - A number by which to multiply the blue channel value.

alphaMult : Number - A number by which to multiply the alpha transparency value.

Example

The following example shows how to merge part of one `BitmapData` with another.

```
fscommand2("SetSoftKeys");

import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
var bitmapData_1:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);
var bitmapData_2:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);
var mc_1:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_1.attachBitmap(bitmapData_1, this.getNextHighestDepth());
var mc_2:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_2.attachBitmap(bitmapData_2, this.getNextHighestDepth());
mc_2._x = 101;

var myListener:Object = new Object ();
myListener.onKeyDown = function () {
    var keyCode = Key.getCode ();
    if (keyCode == ExtendedKey.SOFT1) {
        // Handle left soft key event
        bitmapData_1.merge(bitmapData_2, new Rectangle(0, 0, 50, 40), new Point(25, 20), 128,
0, 0, 0);
    }
};
Key.addListener (myListener);
```

rectangle (BitmapData.rectangle property)

```
public rectangle : Rectangle [read-only]
```

The rectangle that defines the size and location of the bitmap image. The top and left of the rectangle are 0; the width and height are equal to the width and height in pixels of the BitmapData object.

Availability

Flash Lite 3.1

Example

The following example shows that the `rectangle` property of the `Bitmap` instance is read-only by trying to set it and failing:

```
import flash.display.BitmapData;
import flash.geom.Rectangle;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace(myBitmapData.rectangle); // (x=0, y=0, w=100, h=80)

myBitmapData.rectangle = new Rectangle(1, 2, 4, 8);
trace(myBitmapData.rectangle); // (x=0, y=0, w=100, h=80)
```

setPixel (BitmapData.setPixel method)

```
public setPixel(x:Number, y:Number, color:Number) : Void
```

Sets the color of a single pixel of a BitmapData object. The current alpha channel value of the image pixel is preserved during this operation. The value of the RGB color parameter is treated as an unmultiplied color value.

Availability
Flash Lite 3.1

Parameters
x : Number - The *x* position of the pixel whose value changes.

y : Number - The *y* position of the pixel whose value changes.

color : Number - The RGB color to which to set the pixel.

See also
`getPixel (BitmapData.getPixel method)`, `setPixel32 (BitmapData.setPixel32 method)`

setPixel32 (BitmapData.setPixel32 method)

```
public setPixel32(x:Number, y:Number, color:Number) : Void
```

Sets the color and alpha transparency values of a single pixel of a BitmapData object. This method is similar to the `setPixel()` method; the main difference is that the `setPixel32()` method takes an ARGB color value that contains alpha channel information.

Availability
Flash Lite 3.1

Parameters
x : Number - The *x* position of the pixel whose value changes.

y : Number - The *y* position of the pixel whose value changes.

color : Number - The ARGB color to which to set the pixel. If you created an opaque (not a transparent) bitmap, the alpha transparency portion of this color value is ignored.

See also
`getPixel32 (BitmapData.getPixel32 method)`, `setPixel (BitmapData.setPixel method)`

transparent (BitmapData.transparent property)

```
public transparent : Boolean [read-only]
```

Defines whether the bitmap image supports per-pixel transparency. You can set this value only when you construct a BitmapData object by passing in `true` for the `transparent` parameter. After you create a BitmapData object, you can check whether it supports per-pixel transparency by seeing if the value of the `transparent` property is `true`.

Availability
Flash Lite 3.1

width (BitmapData.width property)

```
public width : Number [read-only]
```

The width of the bitmap image in pixels.

Availability
Flash Lite 3.1

Boolean

```
Object
    |
    +-Boolean
```

```
public class Boolean
extends Object
```

The Boolean class is a wrapper object with the same functionality as the standard JavaScript Boolean object. Use the Boolean class to retrieve the primitive data type or string representation of a Boolean object.

You must use the constructor `new Boolean()` to create a Boolean object before calling its methods.

Availability
Flash Lite 2.0

Property summary
Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property) prototype (Object.prototype property), __resolve (Object.__resolve property)

Constructor summary

Signature	Description
Boolean([value:Object])	Creates a Boolean object.

Method summary

Modifiers	Signature	Description
	toString() : String	Returns the string representation (`"true"` or `"false"`) of the Boolean object.
	valueOf() : Boolean	Returns `true` if the primitive value type of the specified Boolean object is true; `false` otherwise.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

Boolean constructor

```
public Boolean([value:Object])
```

Creates a Boolean object. If you omit the `value` parameter, the Boolean object is initialized with a value of `false`. If you specify a value for the `value` parameter, the method evaluates it and returns the result as a Boolean value according to the rules in the global `Boolean()` function.

Availability
Flash Lite 2.0

Parameters
value : Object [optional] - Any expression. The default value is `false`.

Example
The following code creates a new empty Boolean object called `myBoolean`:

```
var myBoolean:Boolean = new Boolean();
```

toString (Boolean.toString method)

```
public toString() : String
```

Returns the string representation (`"true"` or `"false"`) of the Boolean object.

Availability
Flash Lite 2.0

Returns
String - A string; `"true"` or `"false"`.

Example
This example creates a variable of type Boolean and uses `toString()` to convert the value to a string for use in the trace statement:

```
var myBool:Boolean = true;
trace("The value of the Boolean myBool is: " + myBool.toString());
myBool = false;
trace("The value of the Boolean myBool is: " + myBool.toString());
```

valueOf (Boolean.valueOf method)

```
public valueOf() : Boolean
```

Returns `true` if the primitive value type of the specified Boolean object is true; `false` otherwise.

Availability
Flash Lite 2.0

Returns
Boolean - A Boolean value.

Example

The following example shows how this method works, and also shows that the primitive value type of a new Boolean object is `false`:

```
var x:Boolean = new Boolean();
trace(x.valueOf());   // false
x = (6==3+3);
trace(x.valueOf());   // true
```

Button

```
Object
    |
    +-Button
```

```
public class Button
extends Object
```

All button symbols are instances of the Button object. You can give a button an instance name in the Property inspector, and use the methods and properties of the Button class to manipulate buttons with ActionScript. Button instance names are displayed in the Movie Explorer and in the Insert Target Path dialog box in the Actions panel.

Availability

Flash Lite 2.0

See also

Object

Property summary

Modifiers	Property	Description
	_alpha : Number	The alpha transparency value of the button.
	enabled : Boolean	A Boolean value that specifies whether a button is enabled.
	_focusrect : Boolean	A Boolean value that specifies whether a button has a yellow rectangle around it when it has input focus.
	_height : Number	The height of the button, in pixels.
	_highquality : Number	**Deprecated** since Flash Player 7. This property was deprecated in favor of `Button._quality`. Specifies the level of anti-aliasing applied to the current SWF file.
	_name : String	Instance name of the button.
	_parent : MovieClip	A reference to the movie clip or object that contains the current movie clip or object.
	_quality : String	Property (global); sets or retrieves the rendering quality used for a SWF file.
	_rotation : Number	The rotation of the button, in degrees, from its original orientation.

Modifiers	Property	Description
	_soundbuftime : Number	Specifies the number of seconds a sound prebuffers before it starts to stream.
	tabEnabled : Boolean	Specifies whether a button is included in automatic tab ordering.
	tabIndex : Number	Lets you customize the tab ordering of objects in a SWF file.
	_target : String [read-only]	Returns the target path of the button instance.
	trackAsMenu : Boolean	A Boolean value that indicates whether other buttons or movie clips can receive a release event from a mouse or stylus.
	_url : String [read-only]	Retrieves the URL of the SWF file that created the button.
	_visible : Boolean	A Boolean value that indicates whether the button is visible.
	_width : Number	The width of the button, in pixels.
	_x : Number	An integer that sets the x coordinate of a button relative to the local coordinates of the parent movie clip.
	_xmouse : Number [read-only]	Returns the *x* coordinate of the mouse position relative to the button.
	_xscale : Number	The horizontal scale of the button as applied from the registration point of the button, expressed as a percentage.
	_y : Number	The *y* coordinate of the button relative to the local coordinates of the parent movie clip.
	_ymouse : Number [read-only]	Returns the *y* coordinate of the mouse position relative to the button.
	_yscale : Number	The vertical scale of the button as applied from the registration point of the button, expressed as a percentage.

Properties inherited from class Object

```
constructor (Object.constructor property),__proto__ (Object.__proto__
property)prototype (Object.prototype property),__resolve (Object.__resolve property)
```

Event summary

Event	Description
onDragOut = function() {}	Invoked when the user clicks on the button and then drags the pointer outside of the button.
onDragOver = function() {}	Invoked when the user clicks outside of the button and then drags the pointer over the button.
onKeyDown = function() {}	Invoked when a button has keyboard focus and a key is pressed.
onKeyUp = function() {}	Invoked when a button has input focus and a key is released.
onKillFocus = function(newFocus: Object) {}	Invoked when a button loses keyboard focus.
onPress = function() {}	Invoked when a button is pressed.

Event	Description
onRelease = function() {}	Invoked when a button is released.
onReleaseOutside = function() {}	Invoked when the mouse is released with the pointer outside the button after the mouse button is pressed with the pointer inside the button.
onRollOut = function() {}	Invoked when the button loses focus.
onRollOver = function() {}	Invoked when the button gains focus.
onSetFocus = function(oldFocus: Object) {}	Invoked when a button receives keyboard focus.

Method summary

Modifiers	Signature	Description
	getDepth() : Number	Returns the depth of the button instance.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

_alpha (Button._alpha property)

```
public _alpha : Number
```

The alpha transparency value of the button specified by `my_btn`. Valid values are 0 (fully transparent) to 100 (fully opaque). The default value is 100. Objects in a button with `_alpha` set to 0 are active, even though they are invisible.

Availability

Flash Lite 2.0

Example

The following code sets the `_alpha` property of a button named `myBtn_btn` to 50% when the user clicks the button. First, add a Button instance on the Stage. Second, give it an instance name of myBtn_btn. Lastly, with frame 1 selected, place the following code into the Actions panel:

```
myBtn_btn.onRelease = function(){
    this._alpha = 50;
};
```

See also

_alpha (MovieClip._alpha property), _alpha (TextField._alpha property)

enabled (Button.enabled property)

```
public enabled : Boolean
```

A Boolean value that specifies whether a button is enabled. When a button is disabled (the enabled property is set to `false`), the button is visible but cannot be clicked. The default value is `true`. This property is useful if you want to disable part of your navigation; for example, you may want to disable a button in the currently displayed page so that it can't be clicked and the page cannot be reloaded.

Availability
Flash Lite 2.0

Example
The following example demonstrates how you can disable and enable buttons from being clicked. Two buttons, `myBtn1_btn` and `myBtn2_btn`, are on the Stage and the following ActionScript is added so that the `myBtn2_btn` button cannot be clicked. First, add two button instances on the Stage. Second, give them instance names of `myBtn1_btn` and `myBtn2_btn`. Lastly, place the following code on frame 1 to enable or disable buttons.

```
myBtn1_btn.enabled = true;
myBtn2_btn.enabled = false;

//button code
// the following function will not get called
// because myBtn2_btn.enabled was set to false
myBtn1_btn.onRelease = function() {
    trace( "you clicked : " + this._name );
};
myBtn2_btn.onRelease = function() {
    trace( "you clicked : " + this._name );
};
```

_focusrect (Button._focusrect property)

```
public _focusrect : Boolean
```

A Boolean value that specifies whether a button has a yellow rectangle around it when it has input focus. This property can override the global `_focusrect` property. By default, the `_focusrect` property of a button instance is null; the button instance does not override the global `_focusrect` property. If the `_focusrect` property of a button instance is set to `true` or `false`, it overrides the setting of the global `_focusrect` property for the single button instance.

In Flash Player 4 and Flash Player 5 SWF files, the `_focusrect` property controls the global `_focusrect` property. It is a Boolean value. This behavior was changed in Flash Player 6 and later to permit customizing the `_focusrect` property on an individual movie clip.

If the `_focusrect` property is set to `false`, then keyboard navigation for that button is limited to the Tab key. All other keys, including the Enter and arrow keys, are ignored. To restore full keyboard navigation, you must set `_focusrect` to `true`.

Note: For the Flash Lite 2.0 player, when the `_focusrect` property is disabled (in other words, `Button.focusRect` is `false`), the button receives all events. This behavior is different from Flash Lite player behavior because when the `_focusrect` property is disabled, the button receives the `rollOver` and `rollOut` events but does not receive the `press` and `release` events.

Also for Flash Lite 2.0, you can change the color of the focus rectangle using the fscommand2 `SetFocusRectColor` command. This behavior is also different from Flash Lite player, for which the color of the focus rectangle is restricted to yellow.

Availability
Flash Lite 2.0

Example
This example demonstrates how to hide the yellow rectangle around a specified button instance in a SWF file when it has focus in a browser window. Create three buttons called `myBtn1_btn`, `myBtn2_btn`, and `myBtn3_btn`, and add the following ActionScript to Frame 1 of the Timeline:

```
myBtn2_btn._focusrect = false;
```

getDepth (Button.getDepth method)

```
public getDepth() : Number
```

Returns the depth of the button instance.

Each movie clip, button, and text field has a unique depth associated with it that determines how the object appears in front of or in back of other objects. Objects with higher depths appear in front.

Availability
Flash Lite 2.0

Returns
Number - The depth of the button instance.

Example
If you create `myBtn1_btn` and `myBtn2_btn` on the Stage, you can trace their depth using the following ActionScript:

```
trace(myBtn1_btn.getDepth());
trace(myBtn2_btn.getDepth());
```

If you load a SWF file called buttonMovie.swf into this document, you could trace the depth of a button, `myBtn4_btn`, inside that SWF file using another button in the main SWF:

```
this.createEmptyMovieClip("myClip_mc", 999);
myClip_mc.loadMovie("buttonMovie.swf");
myBtn3_btn.onRelease = function(){
    trace(myClip_mc.myBtn4_btn.getDepth());
};
```

You might notice that two of these buttons have the same depth value, one in the main SWF file and one in the loaded SWF file. This is misleading, because buttonMovie.swf was loaded at depth 999, which means that the button it contains will also have a depth of 999 relative to the buttons in the main SWF file. Keep in mind that each movie clip has its own internal z-order, which means that each movie clip has its own set of depth values. The two buttons may have the same depth value, but the values only have meaning in relation to other objects in the same z-order. In this case, the buttons have the same depth value, but the values relate to different movie clips: the depth value of the button in the main SWF file relates to the z-order of the main Timeline, while the depth value of the button in the loaded SWF file relates to the internal z-order of the `myClip_mc` movie clip.

See also
getDepth (MovieClip.getDepth method), getDepth (TextField.getDepth method), getInstanceAtDepth (MovieClip.getInstanceAtDepth method)

_height (Button._height property)

```
public _height : Number
```

The height of the button, in pixels.

Availability

Flash Lite 2.0

Example

The following example sets the height and width of a button called `my_btn` to a specified width and height.

```
my_btn._width = 500;
my_btn._height = 200;
```

_highquality (Button._highquality property)

```
public _highquality : Number
```

Deprecated since Flash Player 7. This property was deprecated in favor of `Button._quality`.

Specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this smooths bitmaps if the SWF file does not contain animation and is the default value. Specify 0 (low quality) to prevent anti-aliasing.

Availability

Flash Lite 2.0

Example

Add a button instance on the Stage and name it myBtn_btn. Draw an oval on the Stage using the Oval tool that has a stroke and fill color. Select Frame 1 and add the following ActionScript using the Actions panel:

```
myBtn_btn.onRelease = function() {
    myBtn_btn._highquality = 0;
};
```

When you click myBtn_btn, the circle's stroke will look jagged. You could add the following ActionScript instead to affect the SWF globally:

```
_quality = 0;
```

See also

_quality (Button._quality property), _quality property

_name (Button._name property)

```
public _name : String
```

Instance name of the button specified by `my_btn`.

Availability

Flash Lite 2.0

Example
The following example traces all instance names of any Button instances within the current Timeline of a SWF file.

```
for (i in this) {
    if (this[i] instanceof Button) {
    trace(this[i]._name);
    }
}
```

onDragOut (Button.onDragOut handler)

```
onDragOut = function() {}
```

Invoked when the user presses the mouse button over the button and then drags the pointer outside of the button. You must define a function that is executed when the event handler is invoked.

Note: The `onDragOut` Event Handler is supported for Flash Lite 2.0 only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
The following example demonstrates how you can execute statements when the pointer is dragged off a button. Create a button called `my_btn` on the Stage and enter the following ActionScript in a frame on the Timeline:

```
my_btn.onDragOut = function() {
    trace("onDragOut: "+this._name);
};
my_btn.onDragOver = function() {
    trace("onDragOver: "+this._name);
};
```

onDragOver (Button.onDragOver handler)

```
onDragOver = function() {}
```

Invoked when the user presses the mouse button outside of the button and then drags the pointer over the button. You must define a function that is executed when the event handler is invoked.

Note: The `onDragOver` Event Handler is supported for Flash Lite 2.0 only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
The following example defines a function for the onDragOver handler that sends a `trace()` statement to the Output panel. Create a button called `my_btn` on the Stage and enter the following ActionScript on the Timeline:

```
my_btn.onDragOut = function() {
    trace("onDragOut: "+this._name);
};
my_btn.onDragOver = function() {
    trace("onDragOver: "+this._name);
};
```

When you test the SWF file, drag the pointer off the button instance. Then, while pressing the mouse button, drag onto the button instance again. Notice that the Output panel tracks your movements.

See also

onDragOut (Button.onDragOut handler)

onKeyDown (Button.onKeyDown handler)

```
onKeyDown = function() {}
```

Invoked when a button has keyboard focus and a key is pressed. The `onKeyDown` event handler is invoked with no parameters. You can use the `Key.getAscii()` and `Key.getCode()` methods to determine which key was pressed. You must define a function that is executed when the event handler is invoked.

Availability

Flash Lite 2.0

Example

In the following example, a function that sends text to the Output panel is defined for the `onKeyDown` handler. Create a button called `my_btn` on the Stage, and enter the following ActionScript in a frame on the Timeline:

```
my_btn.onKeyDown = function() {
    trace("onKeyDown: "+this._name+" (Key: "+getKeyPressed()+")");
};
function getKeyPressed():String {
    var theKey:String;
    switch (Key.getAscii()) {
    case Key.BACKSPACE :
        theKey = "BACKSPACE";
        break;
    case Key.SPACE :
        theKey = "SPACE";
        break;
    default :
        theKey = chr(Key.getAscii());
    }
    return theKey;
}
```

Select Control > Test Movie to test the SWF file. Make sure you select Control > Disable Keyboard Shortcuts in the test environment. Then press the Tab key until the button has focus (a yellow rectangle appears around the `my_btn` instance) and start pressing keys on your keyboard. When you press keys, they are displayed in the Output panel.

See also

onKeyUp (Button.onKeyUp handler), getAscii (Key.getAscii method), getCode (Key.getCode method)

onKeyUp (Button.onKeyUp handler)

```
onKeyUp = function() {}
```

Invoked when a button has input focus and a key is released. The `onKeyUp` event handler is invoked with no parameters. You can use the `Key.getAscii()` and `Key.getCode()` methods to determine which key was pressed.

Availability

Flash Lite 2.0

Example

In the following example, a function that sends text to the Output panel is defined for the `onKeyDown` handler. Create a button called `my_btn` on the Stage, and enter the following ActionScript in a frame on the Timeline:

```
my_btn.onKeyDown = function() {
    trace("onKeyDown: "+this._name+" (Key: "+getKeyPressed()+")");
};
my_btn.onKeyUp = function() {
    trace("onKeyUp: "+this._name+" (Key: "+getKeyPressed()+")");
};
function getKeyPressed():String {
    var theKey:String;
    switch (Key.getAscii()) {
    case Key.BACKSPACE :
        theKey = "BACKSPACE";
        break;
    case Key.SPACE :
        theKey = "SPACE";
        break;
    default :
        theKey = chr(Key.getAscii());
    }
    return theKey;
}
```

Press Control+Enter to test the SWF file. Make sure you select Control > Disable Keyboard Shortcuts in the test environment. Then press the Tab key until the button has focus (a yellow rectangle appears around the `my_btn` instance) and start pressing keys on your keyboard. When you press keys, they are displayed in the Output panel.

See also

`onKeyDown (Button.onKeyDown handler)`, `getAscii (Key.getAscii method)`, `getCode (Key.getCode method)`

onKillFocus (Button.onKillFocus handler)

```
onKillFocus = function(newFocus:Object) {}
```

Invoked when a button loses keyboard focus. The `onKillFocus` handler receives one parameter, `newFocus`, which is an object representing the new object receiving the focus. If no object receives the focus, `newFocus` contains the value `null`.

Availability

Flash Lite 2.0

Parameters

newFocus : Object - The object that is receiving the focus.

Example

The following example demonstrates how statements can be executed when a button loses focus. Create a button instance on the Stage called `my_btn` and add the following ActionScript to Frame 1 of the Timeline:

```
this.createTextField("output_txt", this.getNextHighestDepth(), 0, 0, 300, 200);
output_txt.wordWrap = true;
output_txt.multiline = true;
output_txt.border = true;
my_btn.onKillFocus = function() {
    output_txt.text = "onKillFocus: "+this._name+newline+output_txt.text;
};
```

Test the SWF file in a browser window, and try using the Tab key to move through the elements in the window. When the button instance loses focus, text is sent to the `output_txt` text field.

onPress (Button.onPress handler)

```
onPress = function() {}
```

Invoked when a button is pressed. You must define a function that is executed when the event handler is invoked.

Availability

Flash Lite 2.0

Example

In the following example, a function that sends a `trace()` statement to the Output panel is defined for the `onPress` handler:

```
my_btn.onPress = function () {
    trace ("onPress called");
};
```

onRelease (Button.onRelease handler)

```
onRelease = function() {}
```

Invoked when a button is released. You must define a function that is executed when the event handler is invoked.

Availability

Flash Lite 2.0

Example

In the following example, a function that sends a `trace()` statement to the Output panel is defined for the `onRelease` handler:

```
my_btn.onRelease = function () {
    trace ("onRelease called");
};
```

onReleaseOutside (Button.onReleaseOutside handler)

```
onReleaseOutside = function() {}
```

Invoked when the mouse is released with the pointer outside the button after the mouse button is pressed with the pointer inside the button. You must define a function that is executed when the event handler is invoked.

Note: The `onReleaseOutside` Event Handler is supported for Flash Lite 2.0 only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
In the following example, a function that sends a `trace()` statement to the Output panel is defined for the `onReleaseOutside` handler:

```
my_btn.onReleaseOutside = function () {
    trace ("onReleaseOutside called");
};
```

onRollOut (Button.onRollOut handler)

```
onRollOut = function() {}
```

Invoked when the button loses focus. This can happen when the user clicks another button or area outside of the currently selected button. You must define a function that is executed when the event handler is invoked.

Availability
Flash Lite 2.0

Example
In the following example, a function that sends a `trace()` statement to the Output panel is defined for the `onRollOut` handler:

```
my_btn.onRollOut = function () {
    trace ("onRollOut called");
};
```

onRollOver (Button.onRollOver handler)

```
onRollOver = function() {}
```

Invoked when the button gains focus. This can happen when the user clicks another button outside of the currently selected button. Invoked when the pointer moves over a button area. You must define a function that is executed when the event handler is invoked.

Availability
Flash Lite 2.0

Example
In the following example, a function that sends a `trace()` statement to the Output panel is defined for the `onRollOver` handler:

```
my_btn.onRollOver = function () {
    trace ("onRollOver called");
};
```

onSetFocus (Button.onSetFocus handler)

```
onSetFocus = function(oldFocus:Object) {}
```

Invoked when a button receives keyboard focus. The `oldFocus` parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a text field to a button, `oldFocus` contains the text field instance.

If there is no previously focused object, `oldFocus` contains a null value.

Availability

Flash Lite 2.0

Parameters

oldFocus : Object - The object to lose keyboard focus.

Example

The following example demonstrates how you can execute statements when the user of a SWF file moves focus from one button to another. Create two buttons, `btn1_btn` and `btn2_btn`, and enter the following ActionScript in Frame 1 of the Timeline:

```
Selection.setFocus(btn1_btn);
trace(Selection.getFocus());
btn2_btn.onSetFocus = function(oldFocus) {
    trace(oldFocus._name + "lost focus");
};
```

Test the SWF file by pressing Control+Enter. Make sure you select Control > Disable Keyboard Shortcuts if it is not already selected. Focus is set on `btn1_btn`. When `btn1_btn` loses focus and `btn2_btn` gains focus, information is displayed in the Output panel.

_parent (Button._parent property)

```
public _parent : MovieClip
```

A reference to the movie clip or object that contains the current movie clip or object. The current object is the one containing the ActionScript code that references `_parent`.

Use `_parent` to specify a relative path to movie clips or objects that are above the current movie clip or object. You can use `_parent` to move up multiple levels in the display list as in the following:

```
this._parent._parent._alpha = 20;
```

Availability

Flash Lite 2.0

Example

In the following example, a button named `my_btn` is placed inside a movie clip called `my_mc`. The following code shows how to use the `_parent` property to get a reference to the movie clip `my_mc`:

```
trace(my_mc.my_btn._parent);
```

The Output panel displays the following:

```
_level0.my_mc
```

See also

_parent (MovieClip._parent property), _target (MovieClip._target property), _root property

_quality (Button._quality property)

```
public _quality : String
```

Property (global); sets or retrieves the rendering quality used for a SWF file. Device fonts are always aliased and therefore are unaffected by the _quality property.

The _quality property can be set to the following values:

- "LOW" Low rendering quality. Graphics are not anti-aliased, and bitmaps are not smoothed.
- "MEDIUM" Medium rendering quality. Graphics are anti-aliased using a 2 x 2 pixel grid, but bitmaps are not smoothed. This is suitable for movies that do not contain text.
- "HIGH" High rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are smoothed if the movie is static. This is the default rendering quality setting used by Flash.

Note: Although you can specify this property for a Button object, it is actually a global property, and you can specify its value simply as _quality.

Availability

Flash Lite 2.0

Example

This example sets the rendering quality of a button named my_btn to LOW:

```
my_btn._quality = "LOW";
```

_rotation (Button._rotation property)

```
public _rotation : Number
```

The rotation of the button, in degrees, from its original orientation. Values from 0 to 180 represent clockwise rotation; values from 0 to -180 represent counterclockwise rotation. Values outside this range are added to or subtracted from 360 to obtain a value within the range. For example, the statement my_btn._rotation = 450 is the same as my_btn._rotation = 90.

Availability

Flash Lite 2.0

Example

The following example rotates two buttons on the Stage. Create two buttons on the Stage called control_btn and my_btn. Make sure that my_btn is not perfectly round, so you can see it rotating. Then enter the following ActionScript in Frame 1 of the Timeline:

```
var control_btn:Button;
var my_btn:Button;
control_btn.onRelease = function() {
    my_btn._rotation += 10;
};
```

Now create another button on the Stage called `myOther_btn`, making sure it isn't perfectly round (so you can see it rotate). Enter the following ActionScript in Frame 1 of the Timeline.

```
var myOther_btn:Button;
this.createEmptyMovieClip("rotater_mc", this.getNextHighestDepth());
rotater_mc.onEnterFrame = function() {
    myOther_btn._rotation += 2;
};
```

See also

`_rotation (MovieClip._rotation property)`, `_rotation (TextField._rotation property)`

_soundbuftime (Button._soundbuftime property)

```
public _soundbuftime : Number
```

Specifies the number of seconds a sound prebuffers before it starts to stream.

Note: Although you can specify this property for a Button object, it is actually a global property that applies to all sounds loaded, and you can specify its value simply as `_soundbuftime`. Setting this property for a Button object actually sets the global property.

For more information and an example, see `_soundbuftime`.

Availability

Flash Lite 2.0

See also

`_soundbuftime property`

tabEnabled (Button.tabEnabled property)

```
public tabEnabled : Boolean
```

Specifies whether `my_btn` is included in automatic tab ordering. It is `undefined` by default.

If the `tabEnabled` property is `undefined` or `true`, the object is included in automatic tab ordering. If the `tabIndex` property is also set to a value, the object is included in custom tab ordering as well. If `tabEnabled` is `false`, the object is not included in automatic or custom tab ordering, even if the `tabIndex` property is set.

Availability

Flash Lite 2.0

Example

The following ActionScript is used to set the `tabEnabled` property for one of four buttons to `false`. However, all four buttons (`one_btn`, `two_btn`, `three_btn`, and `four_btn`) are placed in a custom tab order using `tabIndex`. Although `tabIndex` is set for `three_btn`, `three_btn` is not included in a custom or automatic tab order because `tabEnabled` is set to `false` for that instance. To set the tab ordering for the four buttons, add the following ActionScript to Frame 1 of the Timeline:

```
three_btn.tabEnabled = false;
two_btn.tabIndex = 1;
four_btn.tabIndex = 2;
three_btn.tabIndex = 3;
one_btn.tabIndex = 4;
```

See also

`tabIndex (Button.tabIndex property)`, `tabEnabled (MovieClip.tabEnabled property)`, `tabEnabled (TextField.tabEnabled property)`

tabIndex (Button.tabIndex property)

```
public tabIndex : Number
```

Lets you customize the tab ordering of objects in a SWF file. You can set the `tabIndex` property on a button, movie clip, or text field instance; it is `undefined` by default.

If any currently displayed object in the SWF file contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the SWF file. The custom tab ordering only includes objects that have `tabIndex` properties.

The `tabIndex` property may be a non-negative integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` value of `1` precedes an object with a `tabIndex` value of `2`. If two objects have the same `tabIndex` value, the one that precedes the other in the tab ordering is `undefined`.

The custom tab ordering defined by the `tabIndex` property is *flat*. This means that no attention is paid to the hierarchical relationships of objects in the SWF file. All objects in the SWF file with `tabIndex` properties are placed in the tab order, and the tab order is determined by the order of the `tabIndex` values. If two objects have the same `tabIndex` value, the one that goes first is `undefined`. You shouldn't use the same `tabIndex` value for multiple objects.

Availability

Flash Lite 2.0

Example

The following ActionScript is used to set the `tabEnabled` property for one of four buttons to `false`. However, all four buttons (`one_btn`, `two_btn`, `three_btn`, and `four_btn`) are placed in a custom tab order using `tabIndex`. Although `tabIndex` is set for `three_btn`, `three_btn` is not included in a custom or automatic tab order because `tabEnabled` is set to `false` for that instance. To set the tab ordering for the four buttons, add the following ActionScript to Frame 1 of the Timeline:

```
three_btn.tabEnabled = false;
two_btn.tabIndex = 1;
four_btn.tabIndex = 2;
three_btn.tabIndex = 3;
one_btn.tabIndex = 4;
```

See also

tabEnabled (Button.tabEnabled property), tabChildren (MovieClip.tabChildren property), tabEnabled (MovieClip.tabEnabled property), tabIndex (MovieClip.tabIndex property), tabIndex (TextField.tabIndex property)

_target (Button._target property)

```
public _target : String [read-only]
```

Returns the target path of the button instance specified by `my_btn`.

Availability

Flash Lite 2.0

Example

Add a button instance to the Stage with an instance name `my_btn` and add the following code to Frame 1 of the Timeline:

```
trace(my_btn._target); //displays /my_btn
```

Select my_btn and convert it to a movie clip. Give the new movie clip an instance name `my_mc`. Delete the existing ActionScript in Frame 1 of the Timeline and replace it with:

```
my_mc.my_btn.onRelease = function(){
    trace(this._target); //displays /my_mc/my_btn
};
```

To convert the notation from slash notation to dot notation, modify the previous code example to the following:

```
my_mc.my_btn.onRelease = function(){
    trace(eval(this._target)); //displays _level0.my_mc.my_btn
};
```

This lets you access methods and parameters of the target object, such as:

```
my_mc.my_btn.onRelease = function(){
    var target_btn:Button = eval(this._target);
trace(target_btn._name); //displays my_btn
};
```

See also

_target (MovieClip._target property)

trackAsMenu (Button.trackAsMenu property)

```
public trackAsMenu : Boolean
```

A Boolean value that indicates whether other buttons or movie clips can receive a release event from a mouse or stylus. If you drag a stylus or mouse pointer across a button and then release it on a second button, the `onRelease` event is registered for the second button. This allows you to create menus for the second button. You can set the `trackAsMenu` property on any button or movie clip object. If you have not defined the `trackAsMenu` property, the default behavior is `false`.

You can change the `trackAsMenu` property at any time; the modified button immediately takes on the new behavior.

Note: The `trackAsMenu` property is supported for Flash Lite 2.0 only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`..

Availability
Flash Lite 2.0

Example
The following example demonstrates how to track two buttons as a menu. Place two button instances called `one_btn` and `two_btn` on the Stage. Enter the following ActionScript in the Timeline:

```
var one_btn:Button;
var two_btn:Button;
one_btn.trackAsMenu = true;
two_btn.trackAsMenu = true;
one_btn.onRelease = function() {
    trace("clicked one_btn");
};
two_btn.onRelease = function() {
    trace("clicked two_btn");
};
```

To test the SWF file, click the Stage over `one_btn`, hold the mouse button down, and release it over `two_btn`. Then try commenting out the two lines of ActionScript that contain `trackAsMenu` and test the SWF file again to see the difference in button behavior.

See also
`trackAsMenu (MovieClip.trackAsMenu property)`

_url (Button._url property)

```
public _url : String [read-only]
```

Retrieves the URL of the SWF file that created the button.

Availability
Flash Lite 2.0

Example
Create two button instances on the Stage called `one_btn` and `two_btn`. Enter the following ActionScript in Frame 1 of the Timeline:

```
var one_btn:Button;
var two_btn:Button;
this.createTextField("output_txt", 999, 0, 0, 100, 22);
output_txt.autoSize = true;
one_btn.onRelease = function() {
    trace("clicked one_btn");
    trace(this._url);
};
two_btn.onRelease = function() {
    trace("clicked "+this._name);
    var url_array:Array = this._url.split("/");
    var my_str:String = String(url_array.pop());
    output_txt.text = unescape(my_str);
};
```

When you click each button, the file name of the SWF containing the buttons displays in the Output panel.

_visible (Button._visible property)

```
public _visible : Boolean
```

A Boolean value that indicates whether the button specified by `my_btn` is visible. Buttons that are not visible (`_visible` property set to `false`) are disabled.

Availability

Flash Lite 2.0

Example

Create two buttons on the Stage with the instance names `myBtn1_btn` and `myBtn2_btn`. Enter the following ActionScript in Frame 1 of the Timeline:

```
myBtn1_btn.onRelease = function() {
    this._visible = false;
    trace("clicked "+this._name);
};
myBtn2_btn.onRelease = function() {
    this._alpha = 0;
    trace("clicked "+this._name);
};
```

Notice how you can still click `myBtn2_btn` after the alpha is set to 0.

See also

_visible (MovieClip._visible property), _visible (TextField._visible property)

_width (Button._width property)

```
public _width : Number
```

The width of the button, in pixels.

Availability

Flash Lite 2.0

Example

The following example increases the width property of a button called `my_btn`, and displays the width in the Output panel. Enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn.onRelease = function() {
    trace(this._width);
    this._width *= 1.1;
};
```

See also

_width (MovieClip._width property)

_x (Button._x property)

```
public _x : Number
```

An integer that sets the x coordinate of a button relative to the local coordinates of the parent movie clip. If a button is on the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the button is inside a movie clip that has transformations, the button is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90 degrees counterclockwise, the enclosed button inherits a coordinate system that is rotated 90 degrees counterclockwise. The button's coordinates refer to the registration point position.

Availability
Flash Lite 2.0

Example
The following example sets the coordinates of `my_btn` to 0 on the Stage. Create a button called `my_btn` and enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn._x = 0;
my_btn._y = 0;
```

See also
`_xscale (Button._xscale property)`, `_y (Button._y property)`, `_yscale (Button._yscale property)`

_xmouse (Button._xmouse property)

```
public _xmouse : Number [read-only]
```

Returns the *x* coordinate of the mouse position relative to the button.

Note: The `_xmouse` property is supported for Flash Lite 2.0 only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`..

Availability
Flash Lite 2.0

Example
The following example displays the x coordinate of the mouse position for the Stage and a button called `my_btn` that is placed on the Stage. Enter the following ActionScript in Frame 1 of the Timeline:

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var table_str:String = "<textformat tabstops='[50,100]'>";
    table_str += "<b>Stage</b>\t"+"x:"+_xmouse+"\t"+"y:"+_ymouse+newline;
    table_str += "<b>Button</b>\t"+"x:"+my_btn._xmouse+"\t"+"y:"+my_btn._ymouse+newline;
    table_str += "</textformat>";
    mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

See also

_ymouse (Button._ymouse property)

_xscale (Button._xscale property)

```
public _xscale : Number
```

The horizontal scale of the button as applied from the registration point of the button, expressed as a percentage. The default registration point is (0,0).

Scaling the local coordinate system affects the _x and _y property settings, which are defined in pixels. For example, if the parent movie clip is scaled to 50%, setting the _x property moves an object in the button by half the number of pixels that it would if the SWF file were at 100%.

Availability

Flash Lite 2.0

Example

The following example scales a button called my_btn. When you click and release the button, it grows 10% on the *x* and *y* axes. Enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn.onRelease = function(){
    this._xscale *= 1.1;
    this._yscale *= 1.1;
};
```

See also

_x (Button._x property), _y (Button._y property), _yscale (Button._yscale property)

_y (Button._y property)

```
public _y : Number
```

The *y* coordinate of the button relative to the local coordinates of the parent movie clip. If a button is in the main Timeline, its coordinate system refers to the upper left corner of the Stage as (0, 0). If the button is inside another movie clip that has transformations, the button is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90 degrees counterclockwise, the enclosed button inherits a coordinate system that is rotated 90 degrees counterclockwise. The button's coordinates refer to the registration point position.

Availability

Flash Lite 2.0

Example

The following example sets the coordinates of my_btn to 0 on the Stage. Create a button called my_btn and enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn._x = 0;
my_btn._y = 0;
```

See also

_x (Button._x property), _xscale (Button._xscale property), _yscale (Button._yscale property)

_ymouse (Button._ymouse property)

```
public _ymouse : Number [read-only]
```

Returns the *y* coordinate of the mouse position relative to the button.

Note: The `_ymouse` property is supported for Flash Lite 2.0 only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`..

Availability

Flash Lite 2.0

Example

The following example displays the x coordinate of the mouse position for the Stage and a button called `my_btn` that is placed on the Stage. Enter the following ActionScript in Frame 1 of the Timeline:

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var table_str:String = "<textformat tabstops='[50,100]'>";
    table_str += "<b>Stage</b>\t"+"x:"+_xmouse+"\t"+"y:"+_ymouse+newline;
    table_str += "<b>Button</b>\t"+"x:"+my_btn._xmouse+"\t"+"y:"+my_btn._ymouse+newline;
    table_str += "</textformat>";
    mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

See also

`_xmouse (Button._xmouse property)`

_yscale (Button._yscale property)

```
public _yscale : Number
```

The vertical scale of the button as applied from the registration point of the button, expressed as a percentage. The default registration point is (0,0).

Availability

Flash Lite 2.0

Example

The following example scales a button called `my_btn`. When you click and release the button, it grows 10% on the *x* and *y* axes. Enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn.onRelease = function(){
    this._xscale *= 1.1;
    this._yscale *= 1.1;
};
```

See also

_y (Button._y property), _x (Button._x property), _xscale (Button._xscale property)

capabilities (System.capabilities)

```
Object
    |
    +-System.capabilities
```

```
public class capabilities
extends Object
```

The Capabilities class determines the abilities of the system and player that host a SWF file, which lets you tailor content for different formats. For example, the screen of a mobile device is different from a computer screen. To provide appropriate content to as many users as possible, you can use the `System.capabilities` object to determine the type of device a user has. You can then either specify to the server to send different SWF files based on the device capabilities or tell the SWF file to alter its presentation based on the capabilities of the device.

You can send capabilities information using a `GET` or `POST` HTTP method.

The following example shows a string for a mobile device:

- that indicates a normal screen orientation
- that is running an undetermined language
- that is running the Symbian7.0sSeries60V2 operating system
- that is configured so the user can't access hard disk, camera, or microphone
- that has the Flash Lite player as the official release version
- for which the Flash Lite player does not support the development nor playback of screen broadcast applications to be run through Flash Media Server
- that does not support printing on the device
- that the Flash Lite player is running on a mobile device that supports embedded video.

```
undefinedScreenOrientation=normal
language=xu
OS=Symbian7.0sSeries60V2
localFileReadDisable=true
avHardwareDisable=true
isDebugger=false
hasScreenBroadcast=false
hasScreenPlayback=false
hasPrinting=false
hasEmbeddedVideo=true
```

Most properties of the `System.capabilities` object are read-only.

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
static	audioMIMETypes : Array [read-only]	Returns an array of MIME types for audio codecs supported by a mobile device.
static	avHardwareDisable : Boolean [read-only]	A Boolean value that specifies whether access to the user's camera and microphone has been administratively prohibited (true) or allowed (false).
static	has4WayKeyAS : Boolean [read-only]	A Boolean value that is true if the Flash Lite player executes the ActionScript code associated with key event handlers that are associated with the Left, Right, Up, and Down keys.
static	hasAccessibility : Boolean [read-only]	A Boolean value that is true if the player is running in an environment that supports communication between Flash Lite player and accessibility aids; false otherwise.
static	hasAudio : Boolean [read-only]	Specifies if the system has audio capabilities.
static	hasAudioEncoder : Boolean [read-only]	Specifies if the Flash Lite player can encode an audio stream.
static	hasCMIDI : Boolean [read-only]	Returns true if the mobile device can play sound data in the CMIDI audio format.
static	hasCompoundSound : Boolean [read-only]	Returns true if the Flash Lite player can process compound sound data.
static	hasDataLoading : Boolean [read-only]	Returns true if the Flash Lite player can dynamically load additional data through calls to specific functions.
static	hasEmail : Boolean [read-only]	Returns true if the Flash Lite player can send e-mail messages with the GetURL ActionScript command.
static	hasEmbeddedVideo : Boolean [read-only]	A Boolean value that indicates whether the mobile device supports embedded video.
static	hasMappableSoftKeys : Boolean	Returns true if the mobile device allows you to reset or reassign softkey labels and handle events from those softkeys.
static	hasMFI : Boolean [read-only]	Returns true if the mobile device is capable of playing sound data in the MFI audio format.
static	hasMIDI : Boolean [read-only]	Returns true if the mobile device is capable of playing sound data in the MIDI audio format.
static	hasMMS : Boolean [read-only]	Returns true if the mobile device can send MMS messages with the GetURL ActionScript command.
static	hasMouse : Boolean [read-only]	Indicates whether the mobile device sends mouse-related events to a Flash Lite player.
static	hasMP3 : Boolean [read-only]	Specifies if the mobile device has a MP3 decoder.
static	hasPrinting : Boolean [read-only]	A Boolean value that is true if the player is running on a mobile device that supports printing; false otherwise.
static	hasQWERTYKeyboard : Boolean [read-only]	Returns true if the Flash Lite player can process ActionScript code associated with all keys found on a standard QWERTY keyboard, including the BACKSPACE key.
static	hasScreenBroadcast : Boolean [read-only]	A Boolean value that is true if the player supports the development of screen broadcast applications to be run through Flash Media Server; false otherwise.

Modifiers	Property	Description
static	hasScreenPlayback : Boolean [read-only]	A Boolean value that is true if the player supports the playback of screen broadcast applications that are being run through Flash Media Server; false otherwise.
static	hasSharedObjects : Boolean [read-only]	Returns true if the Flash Lite content playing back in an application can access the Flash Lite version of shared objects.
static	hasSMAF : Boolean [read-only]	Returns true if the mobile device is capable of playing sound data in the SMAF audio format.
static	hasSMS : Number [read-only]	Indicates whether the mobile device can send SMS messages with the GetURL ActionScript command.
static	hasStreamingAudio : Boolean [read-only]	A Boolean value that is true if the player can play streaming audio; false otherwise.
static	hasStreamingVideo : Boolean [read-only]	A Boolean value that indicates whether the player can play streaming video.
static	hasStylus : Boolean [read-only]	Indicates if the mobile device supports stylus-related events.
static	hasVideoEncoder : Boolean [read-only]	Specifies if the Flash Lite player can encode a video stream.
static	hasXMLSocket : Number [read-only]	Indicates whether the host application supports XML sockets.
static	imageMIMETypes : Array [read-only]	Returns an array that contains all MIME types that the loadMovie function and the codecs for a mobile device support for processing images.
static	isDebugger : Boolean [read-only]	A Boolean value that indicates whether the player is an officially released version (false) or a special debugging version (true).
static	language : String [read-only]	Indicates the language of the system on which the player is running.
static	localFileReadDisable : Boolean [read-only]	A Boolean value that indicates whether read access to the user's hard disk has been administratively prohibited (true) or allowed (false).
static	MIMETypes : Array [read-only]	Returns an array that contains all MIME types that the loadMovie function, Sound and Video objects support.
static	os : String [read-only]	A string that indicates the current operating system.
static	screenOrientation : String [read-only]	This member variable of the System.capabilities object that indicates the current screen orientation.
static	screenResolutionX : Number [read-only]	An integer that indicates the maximum horizontal resolution of the screen.
static	screenResolutionY : Number [read-only]	An integer that indicates the maximum vertical resolution of the screen.
static	softKeyCount : Number [read-only]	Indicates the number of remappable soft keys that the mobile device supports.
static	version : String [read-only]	A string that contains the Flash Lite player platform and version information (for example, "WIN 7,1,0,0").
static	videoMIMETypes : Array [read-only]	Indicates all the MIME types for video that the mobile device's codecs support.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property) prototype (Object.prototype property), __resolve (Object.__resolve property)

Method summary

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

audioMIMETypes (capabilities.audioMIMETypes property)

```
public static audioMIMETypes : Array [read-only]
```

Returns an array of MIME types for audio codecs supported by a mobile device.

Availability

Flash Lite 2.0

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.audioMIMETypes);
```

avHardwareDisable (capabilities.avHardwareDisable property)

```
public static avHardwareDisable : Boolean [read-only]
```

A Boolean value that specifies whether access to the user's camera and microphone has been administratively prohibited (`true`) or allowed (`false`). The server string is `AVD`.

Note: For Flash Lite 2.0, the value returned is always `true`.

Availability

Flash Lite 2.0

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.avHardwareDisable);
```

has4WayKeyAS (capabilities.has4WayKeyAS property)

```
public static has4WayKeyAS : Boolean [read-only]
```

A Boolean value that is `true` if the Flash Lite player executes the ActionScript code associated with key event handlers that are associated with the Left, Right, Up, and Down keys. Otherwise, this property returns `false`.

If the value of this variable is `true`, when one of the four-way keys is pressed, the player first looks for a handler for that key. If none is found, Flash performs control navigation. However, if an event handler is found, no navigation action occurs for that key. In other words, the presence of a keypress handler for a down key disables the ability to navigate down.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.has4WayKeyAS);
```

hasAccessibility (capabilities.hasAccessibility property)

```
public static hasAccessibility : Boolean [read-only]
```

A Boolean value that is `true` if the player is running in an environment that supports communication between Flash Lite player and accessibility aids; `false` otherwise. The server string is `ACC`.

Note: For Flash Lite 2.0, the value returned is always `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasAccessibility);
```

hasAudio (capabilities.hasAudio property)

```
public static hasAudio : Boolean [read-only]
```

Specifies if the system has audio capabilities. A Boolean value that is `true` if the player is running on a system that has audio capabilities; `false` otherwise. The server string is `A`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasAudio);
```

hasAudioEncoder (capabilities.hasAudioEncoder property)

```
public static hasAudioEncoder : Boolean [read-only]
```

Specifies if the Flash Lite player can encode an audio stream. A Boolean value that is `true` if the player can encode an audio stream, such as that coming from a microphone; `false` otherwise. The server string is `AE`.

Note: For Flash Lite 2.0, the value returned is always `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasAudioEncoder);
```

hasCMIDI (capabilities.hasCMIDI property)

```
public static hasCMIDI : Boolean [read-only]
```

Returns `true` if the mobile device can play sound data in the CMIDI audio format. Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasCMIDI);
```

hasCompoundSound (capabilities.hasCompoundSound property)

```
public static hasCompoundSound : Boolean [read-only]
```

Returns `true` if the Flash Lite player can process compound sound data. Otherwise, it returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasCompoundSound);
```

hasDataLoading (capabilities.hasDataLoading property)

```
public static hasDataLoading : Boolean [read-only]
```

Returns `true` if the Flash Lite player can dynamically load additional data through calls to specific functions.

You can call the following specific functions:

- `loadMovie()`
- `loadMovieNum()`
- `loadVariables()`
- `loadVariablesNum()`
- `XML.parseXML()`
- `Sound.loadSound()`
- `MovieClip.loadVariables()`
- `MovieClip.loadMovie()`
- `MovieClipLoader.loadClip()`
- `LoadVars.load()`
- `LoadVars.sendAndLoad()`

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasDataLoading);
```

hasEmail (capabilities.hasEmail property)

```
public static hasEmail : Boolean [read-only]
```

Returns `true` if the Flash Lite player can send e-mail messages with the `GetURL` ActionScript command.

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasEmail);
```

hasEmbeddedVideo (capabilities.hasEmbeddedVideo property)

```
public static hasEmbeddedVideo : Boolean [read-only]
```

A Boolean value that indicates whether the mobile device supports embedded video.

Note: The `hasEmbeddedVideo` property is always `true` in Flash Lite 2.0 and Flash Lite 2.1, indicating library support for device video.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasEmbeddedVideo);
```

hasMappableSoftKeys (capabilities.hasMappableSoftKeys property)

```
public static hasMappableSoftKeys : Boolean
```

Returns `true` if the mobile device allows you to reset or reassign softkey labels and handle events from those softkeys. Otherwise, `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasMappableSoftKeys);
```

hasMFI (capabilities.hasMFI property)

```
public static hasMFI : Boolean [read-only]
```

Returns `true` if the mobile device is capable of playing sound data in the MFI audio format.

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasMFI);
```

hasMIDI (capabilities.hasMIDI property)

```
public static hasMIDI : Boolean [read-only]
```

Returns `true` if the mobile device is capable of playing sound data in the MIDI audio format.

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasMIDI);
```

hasMMS (capabilities.hasMMS property)

```
public static hasMMS : Boolean [read-only]
```

Returns `true` if the mobile device can send MMS messages with the `GetURL` ActionScript command.

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasMMS);
```

hasMouse (capabilities.hasMouse property)

```
public static hasMouse : Boolean [read-only]
```

Indicates whether the mobile device sends mouse-related events to a Flash Lite player.

This property returns `true` if the mobile device sends mouse-related events to a Flash Lite player. Otherwise, it returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasMouse);
```

hasMP3 (capabilities.hasMP3 property)

```
public static hasMP3 : Boolean [read-only]
```

Specifies if the mobile device has an MP3 decoder. A Boolean value that is `true` if the player is running on a system that has an MP3 decoder; `false` otherwise. The server string is `MP3`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasMP3);
```

hasPrinting (capabilities.hasPrinting property)

```
public static hasPrinting : Boolean [read-only]
```

A Boolean value that is `true` if the player is running on a mobile device that supports printing; `false` otherwise. The server string is `PR`.

Note: For Flash Lite 2.0, the value returned is always `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasPrinting);
```

hasQWERTYKeyboard (capabilities.hasQWERTYKeyboard property)

```
public static hasQWERTYKeyboard : Boolean [read-only]
```

Returns `true` if the Flash Lite player can process ActionScript code associated with all keys found on a standard QWERTY keyboard, including the BACKSPACE key.

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasQWERTYKeyboard);
```

hasScreenBroadcast (capabilities.hasScreenBroadcast property)

```
public static hasScreenBroadcast : Boolean [read-only]
```

A Boolean value that is `true` if the player supports the development of screen broadcast applications to be run through Flash Media Server; `false` otherwise. The server string is `SB`.

Note: For Flash Lite 2.0, the value returned is always `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasScreenBroadcast);
```

hasScreenPlayback (capabilities.hasScreenPlayback property)

```
public static hasScreenPlayback : Boolean [read-only]
```

A Boolean value that is `true` if the player supports the playback of screen broadcast applications that are being run through Flash Media Server; `false` otherwise. The server string is `SP`.

Note: For Flash Lite 2.0, the value returned is always `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasScreenPlayback);
```

hasSharedObjects (capabilities.hasSharedObjects property)

```
public static hasSharedObjects : Boolean [read-only]
```

Returns `true` if the Flash Lite content playing back in an application can access the Flash Lite version of shared objects.

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasSharedObjects);
```

hasSMAF (capabilities.hasSMAF property)

```
public static hasSMAF : Boolean [read-only]
```

Returns `true` if the mobile device is capable of playing sound data in the SMAF audio format.

Otherwise, this property returns `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasSMAF);
```

hasSMS (capabilities.hasSMS property)

```
public static hasSMS : Number [read-only]
```

Indicates whether the mobile device can send SMS messages with the `GetURL` ActionScript command.

If Flash Lite can send SMS messages, this variable is defined and has a value of 1. Otherwise, this variable is not defined.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasSMS);
```

hasStreamingAudio (capabilities.hasStreamingAudio property)

```
public static hasStreamingAudio : Boolean [read-only]
```

A Boolean value that is `true` if the player can play streaming audio; `false` otherwise. The server string is `SA`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasStreamingAudio);
```

hasStreamingVideo (capabilities.hasStreamingVideo property)

```
public static hasStreamingVideo : Boolean [read-only]
```

A Boolean value that indicates whether the player can play streaming video.

Note: The `hasStreamingVideo` property is always `false` in Flash Lite 2.0 and Flash Lite 2.1, indicating that streaming FLV is not supported.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasStreamingVideo);
```

hasStylus (capabilities.hasStylus property)

```
public static hasStylus : Boolean [read-only]
```

Indicates if the mobile device supports stylus-related events.

This property returns `true` if the platform for the mobile device does not support stylus-related events. Otherwise, this property returns `false`.

The stylus does not support the onMouseMove event. This capabilities flag allows the Flash content to check if the platform for a mobile device supports this event.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasStylus);
```

hasVideoEncoder (capabilities.hasVideoEncoder property)

```
public static hasVideoEncoder : Boolean [read-only]
```

Specifies if the Flash Lite player can encode a video stream. A Boolean value that is `true` if the player can encode a video stream, such as that coming from a web camera; `false` otherwise. The server string is `VE`.

Note: For Flash Lite 2.0, the value returned is always `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.hasVideoEncoder);
```

hasXMLSocket (capabilities.hasXMLSocket property)

```
public static hasXMLSocket : Number [read-only]
```

Indicates whether the host application supports XML sockets.

If the host application supports XML sockets, this variable is defined and has a value of 1. Otherwise, this variable is not defined.

imageMIMETypes (capabilities.imageMIMETypes property)

```
public static imageMIMETypes : Array [read-only]
```

Returns an array that contains all MIME types that the `loadMovie` function and the codecs for a mobile device support for processing images.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.imageMIMETypes);
```

isDebugger (capabilities.isDebugger property)

```
public static isDebugger : Boolean [read-only]
```

A Boolean value that indicates whether the player is an officially released version (`false`) or a special debugging version (`true`). The server string is `DEB`.

Note: For Flash Lite 2.0, the value returned is always `false`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.isDebugger);
```

```
***note about space instead of tab used for indents in code
```

language (capabilities.language property)

```
public static language : String [read-only]
```

Indicates the language of the system on which the player is running. This property is specified as a lowercase two-letter language code from ISO 639-1. For Chinese, an additional uppercase two-letter country code subtag from ISO 3166 distinguishes between Simplified and Traditional Chinese. The languages themselves are named with the English tags. For example, `fr` specifies French.

This property changed in two ways for Flash Player 7. First, the language code for English systems no longer includes the country code. In Flash Player 6, all English systems return the language code and the two-letter country code subtag (`en-US`). In Flash Player 7, English systems return only the language code (`en`). Second, on Microsoft Windows systems this property now returns the User Interface (UI) Language. In Flash Player 6 on the Microsoft Windows platform, System.capabilities.language returns the User Locale, which controls settings for formatting dates, times, currency, and large numbers. In Flash Player 7 on the Microsoft Windows platform, this property now returns the UI Language, which refers to the language used for all menus, dialog boxes, error messages, and help files.

Language	**Tag**
Czech	cs
Danish	da
Dutch	nl
English	en
Finnish	fi
French	fr
German	de
Hungarian	hu
Italian	it
Japanese	ja
Korean	ko
Norwegian	no
Other/unknown	xu
Polish	pl
Portuguese	pt
Russian	ru
Simplified Chinese	zh-CN
Spanish	es
Swedish	sv
Traditional Chinese	zh-TW
Turkish	tr

Availability

Flash Lite 2.0

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.language);
```

localFileReadDisable (capabilities.localFileReadDisable property)

```
public static localFileReadDisable : Boolean [read-only]
```

A Boolean value that indicates whether read access to the user's hard disk has been administratively prohibited (`true`) or allowed (`false`). If set to `true`, Flash Lite player will be unable to read files (including the first SWF file that Flash Lite player launches with) from the user's hard disk. For example, attempts to read a file on the user's hard disk using `XML.load()`, `LoadMovie()`, or `LoadVars.load()` will fail if this property is set to `true`.

Reading runtime shared libraries will also be blocked if this property is set to `true`, but reading local shared objects is allowed without regard to the value of this property. The server string is `LFD`.

Note: For Flash Lite 2.0, the value returned is always `true`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.localFileReadDisable);
```

MIMETypes (capabilities.MIMETypes property)

```
public static MIMETypes : Array [read-only]
```

Returns an array that contains all MIME types that the `loadMovie` function, Sound and Video objects support.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.MIMETypes);
```

os (capabilities.os property)

```
public static os : String [read-only]
```

A string that indicates the current operating system. The `os` property can return the following strings: `"Windows XP"`, `"Windows 2000"`, `"Windows NT"`, `"Windows 98/ME"`, `"Windows 95"`, `"Windows CE"` (available only in Flash Player SDK, not in the desktop version), `"Linux"`, and `"MacOS"`. The server string is `OS`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.os);
```

screenOrientation (capabilities.screenOrientation property)

```
public static screenOrientation : String [read-only]
```

A member variable of the `System.capabilities` object that indicates the current screen orientation.

Possible values for `screenOrientation` property:

- `normal` the screen is in its normal orientation
- `rotated90` the screen is rotated by 90 degrees
- `rotated180` the screen is rotated by 180 degrees
- `rotated270` the screen is rotated by 270 degrees

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.screenOrientation);
```

screenResolutionX (capabilities.screenResolutionX property)

```
public static screenResolutionX : Number [read-only]
```

An integer that indicates the maximum horizontal resolution of the screen. The server string is `R` (which returns both the width and height of the screen).

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.screenResolutionX);
```

screenResolutionY (capabilities.screenResolutionY property)

```
public static screenResolutionY : Number [read-only]
```

An integer that indicates the maximum vertical resolution of the screen. The server string is `R` (which returns both the width and height of the screen).

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.screenResolutionY);
```

softKeyCount (capabilities.softKeyCount property)

```
public static softKeyCount : Number [read-only]
```

Indicates the number of remappable soft keys that the mobile device supports.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.softKeyCount);
```

version (capabilities.version property)

```
public static version : String [read-only]
```

A string that contains the Flash Lite player platform and version information (for example, `"WIN 7,1,0,0"`). The server string is `V`.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.version);
```

videoMIMETypes (capabilities.videoMIMETypes property)

```
public static videoMIMETypes : Array [read-only]
```

Indicates all the MIME types for video that the mobile device's codecs support.

This property returns an array of all the MIME types for video that the mobile device's codecs support.

Availability
Flash Lite 2.0

Example
The following example traces the value of this read-only property:

```
trace(System.capabilities.videoMIMETypes);
```

Color

```
Object
    |
    +-Color
```

```
public class Color
extends Object
```

The Color class lets you set the RGB color value and color transform of movie clips and retrieve those values once they have been set.

You must use the constructor `new Color()` to create a Color object before calling its methods.

Availability

Flash Lite 2.0

Property summary

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property) prototype (Object.prototype property), __resolve (Object.__resolve property)

Constructor summary

Signature	Description
Color(target:Object)	Creates a Color object for the movie clip specified by the target_mc parameter.

Method summary

Modifiers	Signature	Description
	getRGB() : Number	Returns the R+G+B combination currently in use by the color object.
	getTransform() : Object	Returns the transform value set by the last Color.setTransform() call.
	setRGB(offset:Number) : Void	Specifies an RGB color for a Color object.
	setTransform(transformObject:Object) : Void	Sets color transform information for a Color object.

Methods inherited from class Object

addProperty (Object.addProperty method),hasOwnProperty (Object.hasOwnProperty method)isPropertyEnumerable (Object.isPropertyEnumerable method)isPrototypeOf (Object.isPrototypeOf method)registerClass (Object.registerClass method),toString (Object.toString method)unwatch (Object.unwatch method),valueOf (Object.valueOf method)watch (Object.watch method)

Color constructor

```
public Color(target:Object)
```

Creates a Color object for the movie clip specified by the `target_mc` parameter. You can then use the methods of that Color object to change the color of the entire target movie clip.

Availability

Flash Lite 2.0

Parameters

target : Object - The instance name of a movie clip.

Example

The following example creates a Color object called `my_color` for the movie clip `my_mc` and sets its RGB value to orange:

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xff9933);
```

getRGB (Color.getRGB method)

```
public getRGB() : Number
```

Returns the R+G+B combination currently in use by the color object.

Availability

Flash Lite 2.0

Returns

Number - A number that represents the RGB numeric value for the color specified.

Example

The following code retrieves the RGB value for the Color object `my_color`, converts the value to a hexadecimal string, and assigns it to the `myValue` variable. To see this code work, add a movie clip instance to the Stage, and give it the instance name `my_mc`:

```
var my_color:Color = new Color(my_mc);
// set the color
my_color.setRGB(0xff9933);
var myValue:String = my_color.getRGB().toString(16);
// trace the color value
trace(myValue); // traces ff9933
```

See also

setRGB (Color.setRGB method)

getTransform (Color.getTransform method)

```
public getTransform() : Object
```

Returns the transform value set by the last `Color.setTransform()` call.

Availability

Flash Lite 2.0

Returns

Object - An object whose properties contain the current offset and percentage values for the specified color.

Example

The following example gets the transform object, and then sets new percentages for colors and alpha of `my_mc` relative to their current values. To see this code work, place a multicolored movie clip on the Stage with the instance name `my_mc`. Then place the following code on Frame 1 in the main Timeline and select Control > Test Movie:

```
var my_color:Color = new Color(my_mc);
var myTransform:Object = my_color.getTransform();
myTransform = { ra: 50, ba: 50, aa: 30};
my_color.setTransform(myTransform);
```

For descriptions of the parameters for a color transform object, see `Color.setTransform()`.

See also

setTransform (Color.setTransform method)

setRGB (Color.setRGB method)

```
public setRGB(offset:Number) : Void
```

Specifies an RGB color for a Color object. Calling this method overrides any previous `Color.setTransform()` settings.

Availability

Flash Lite 2.0

Parameters

offset : Number - `0xRRGGBB` The hexadecimal or RGB color to be set. `RR`, `GG`, and `BB` each consist of two hexadecimal digits that specify the offset of each color component. The `0x` tells the ActionScript compiler that the number is a hexadecimal value.

Example

This example sets the RGB color value for the movie clip `my_mc`. To see this code work, place a movie clip on the Stage with the instance name `my_mc`. Then place the following code on Frame 1 in the main Timeline and select Control > Test Movie:

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xFF0000); // my_mc turns red
```

See also

setTransform (Color.setTransform method)

setTransform (Color.setTransform method)

```
public setTransform(transformObject:Object) : Void
```

Sets color transform information for a Color object. The *colorTransformObject* parameter is a generic object that you create from the `new Object` constructor. It has parameters specifying the percentage and offset values for the red, green, blue, and alpha (transparency) components of a color, entered in the format 0xRRGGBBAA.

The parameters for a color transform object correspond to the settings in the Advanced Effect dialog box and are defined as follows:

- *ra* is the percentage for the red component (-100 to 100).
- *rb* is the offset for the red component (-255 to 255).
- *ga* is the percentage for the green component (-100 to 100).
- *gb* is the offset for the green component (-255 to 255).
- *ba* is the percentage for the blue component (-100 to 100).
- *bb* is the offset for the blue component (-255 to 255).
- *aa* is the percentage for alpha (-100 to 100).
- *ab* is the offset for alpha (-255 to 255).

You create a *colorTransformObject* parameter as follows:

```
var myColorTransform:Object = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

You can also use the following syntax to create a *colorTransformObject* parameter:

```
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb: 90, aa: 40, ab: 70}
```

Availability

Flash Lite 2.0

Parameters

transformObject : Object - An object created with the `new Object` constructor. This instance of the Object class must have the following properties that specify color transform values: `ra`, `rb`, `ga`, `gb`, `ba`, `bb`, `aa`, `ab`. These properties are explained below.

Example

This example creates a new Color object for a target SWF file, creates a generic object called `myColorTransform` with the properties defined above, and uses the `setTransform()` method to pass the *colorTransformObject* to a Color object. To use this code in a Flash (FLA) document, place it on Frame 1 on the main Timeline and place a movie clip on the Stage with the instance name `my_mc`, as in the following code:

```
// Create a color object called my_color for the target my_mc
var my_color:Color = new Color(my_mc);
// Create a color transform object called myColorTransform using
// Set the values for myColorTransform
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb: 90, aa: 40, ab:
70};
// Associate the color transform object with the Color object
// created for my_mc
my_color.setTransform(myColorTransform);
```

See also

Object

ColorTransform (flash.geom.ColorTransform)

```
Object
    |
    +-flash.geom.ColorTransform

public class ColorTransform
extends Object
```

The ColorTransform class lets you mathematically adjust all of the color values in a movie clip. The color adjustment function or *color transformation* can be applied to all four channels: red, green, blue, and alpha transparency.

When a ColorTransform object is applied to a movie clip, a new value for each color channel is calculated like this:

- New red value = (old red value * `redMultiplier)` + `redOffset`
- New green value = (old green value * `greenMultiplier)` + `greenOffset`
- New blue value = (old blue value * `blueMultiplier)` + `blueOffset`
- New alpha value = (old alpha value * `alphaMultiplier)` + `alphaOffset`

If any of the color channel values is greater than 255 after the calculation, it is set to 255. If it is less than 0, it is set to 0.

You must use the `new ColorTransform()` constructor to create a ColorTransform object before you can call the methods of the ColorTransform object.

Color transformations do not apply to the background color of a movie clip (such as a loaded SWF object). They apply only to graphics and symbols that are attached to the movie clip.

Availability

Flash Lite 3.1

See also

colorTransform (Transform.colorTransform property)

Property summary

Modifiers	Property	Description
	alphaMultiplier : Number	A decimal value that is multiplied by the alpha transparency channel value.
	alphaOffset : Number	A number from -255 to 255 that is added to the alpha transparency channel value after it has been multiplied by the `alphaMultiplier` value.
	blueMultiplier : Number	A decimal value that is multiplied by the blue channel value.
	blueOffset : Number	A number from -255 to 255 that is added to the blue channel value after it has been multiplied by the `blueMultiplier` value.
	greenMultiplier : Number	A decimal value that is multiplied by the green channel value.
	greenOffset : Number	A number from -255 to 255 that is added to the green channel value after it has been multiplied by the `greenMultiplier` value.
	redMultiplier : Number	A decimal value that is multiplied by the red channel value.
	redOffset : Number	A number from -255 to 255 that is added to the red channel value after it has been multiplied by the `redMultiplier` value.
	rgb : Number	The RGB color value for a ColorTransform object.

"`constructor (Object.constructor property)`" on page 496, `__proto__ (Object.__proto__ property)`,`prototype (Object.prototype property)`, "`__resolve (Object.__resolve property)`" on page 501

Constructor summary

Signature	Description
ColorTransform ([redMultiplier : Number] , [greenMultiplier : Number] , [blueMultiplier : Number] , [alphaMultiplier : Number] , [redOffset : Number] , [greenOffset : Number] , [blueOffset : Number] , [alphaOffset : Number])	Creates a ColorTransform object for a display object with the specified color channel values and alpha values.

Method summary

Modifiers	Signature	Description
	concat `(second:ColorTransform) : Void`	Applies a second, additive color transformation to the movie clip.
	toString `() : String`	Formats and returns a string that describes all of the properties of the ColorTransform object.

"addProperty (Object.addProperty method)" on page 494, "hasOwnProperty (Object.hasOwnProperty method)" on page 497, "isPropertyEnumerable (Object.isPropertyEnumerable method)" on page 497, "isPrototypeOf (Object.isPrototypeOf method)" on page 498, "registerClass (Object.registerClass method)" on page 500, "toString (Object.toString method)" on page 504, "unwatch (Object.unwatch method)" on page 505, valueOf (Object.valueOf method), "watch (Object.watch method)" on page 507

alphaMultiplier (ColorTransform.alphaMultiplier property)

```
public alphaMultiplier : Number
```

A decimal value that is multiplied by the alpha transparency channel value.

If you set the alpha transparency value of a movie clip directly by using the `MovieClip._alpha` property, it affects the value of the `alphaMultiplier` property of that movie clip's ColorTransform object.

Availability

Flash Lite 3.1

Example

The following example creates the ColorTransform object `colorTrans` and adjusts its `alphaMultiplier` value from 1 to .5.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.alphaMultiplier); // 1

colorTrans.alphaMultiplier = .5;
trace(colorTrans.alphaMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

See also

_alpha (MovieClip._alpha property)

alphaOffset (ColorTransform.alphaOffset property)

```
public alphaOffset : Number
```

A number from -255 to 255 that is added to the alpha transparency channel value after it has been multiplied by the `alphaMultiplier` value.

Availability
Flash Lite 3.1

Example
The following example creates the ColorTransform object `colorTrans` and adjusts its `alphaOffset` value from 0 to -128.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.alphaOffset); // 0

colorTrans.alphaOffset = -128;
trace(colorTrans.alphaOffset); // -128

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

blueMultiplier (ColorTransform.blueMultiplier property)

```
public blueMultiplier : Number
```

A decimal value that is multiplied by the blue channel value.

Availability
Flash Lite 3.1

Example
The following example creates the ColorTransform object `colorTrans` and adjusts its `blueMultiplier` value from 1 to .5.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.blueMultiplier); // 1

colorTrans.blueMultiplier = .5;
trace(colorTrans.blueMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0x0000FF);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

blueOffset (ColorTransform.blueOffset property)

```
public blueOffset : Number
```

A number from -255 to 255 that is added to the blue channel value after it has been multiplied by the `blueMultiplier` value.

Availability

Flash Lite 3.1

Example

The following example creates the ColorTransform object `colorTrans` and adjusts its `blueOffset` value from 0 to 255.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.blueOffset); // 0

colorTrans.blueOffset = 255;
trace(colorTrans.blueOffset); // 255

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

ColorTransform constructor

```
public ColorTransform([redMultiplier:Number], [greenMultiplier:Number],
[blueMultiplier:Number], [alphaMultiplier:Number], [redOffset:Number], [greenOffset:Number],
[blueOffset:Number], [alphaOffset:Number])
```

Creates a ColorTransform object for a display object with the specified color channel values and alpha values.

Availability

Flash Lite 3.1

Parameters

redMultiplier : Number [optional] - The value for the red multiplier, in the range from 0 to 1. The default value is 1.

greenMultiplier : Number [optional] - The value for the green multiplier, in the range from 0 to 1. The default value is 1.

blueMultiplier : Number [optional] - The value for the blue multiplier, in the range from 0 to 1. The default value is 1.

alphaMultiplier : Number [optional] - The value for the alpha transparency multiplier, in the range from 0 to 1. The default value is 1.

redOffset : Number [optional] - The offset for the red color channel value (-255 to 255). The default value is 0.

greenOffset : Number [optional] - The offset for the green color channel value (-255 to 255). The default value is 0.

blueOffset : Number [optional] - The offset for the blue color channel value (-255 to 255). The default value is 0.

alphaOffset : Number [optional] - The offset for alpha transparency channel value (-255 to 255). The default value is 0.

Example

The following example creates a ColorTransform object called `greenTransform`:

```
var greenTransform:flash.geom.ColorTransform = new flash.geom.ColorTransform(0.5, 1.0, 0.5,
0.5, 10, 10, 10, 0);
```

The following example creates the ColorTransform object `colorTrans_1` with the default constructor values. The fact that `colorTrans_1` and `colorTrans_2` trace the same values is evidence that the default constructor values are used.

```
import flash.geom.ColorTransform;

var colorTrans_1:ColorTransform = new ColorTransform(1, 1, 1, 1, 0, 0, 0, 0);
trace(colorTrans_1);
//(redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=0,
greenOffset=0, blueOffset=0, alphaOffset=0)

var colorTrans_2:ColorTransform = new ColorTransform();
trace(colorTrans_2);
//(redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=0,
greenOffset=0, blueOffset=0, alphaOffset=0)
```

concat (ColorTransform.concat method)

```
public concat(second:ColorTransform) : Void
```

Applies a second, additive color transformation to the movie clip. The second set of transformation parameters is applied to the colors of the movie clip after the first transformation has been completed.

Availability

Flash Lite 3.1

Parameters

second : ColorTransform - A second ColorTransform object to be combined with the current ColorTransform object.

Example

The following example concatenates the ColorTransform object `colorTrans_2` to `colorTrans_1` resulting in a full red offset combined with a .5 alpha multiplier.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans_1:ColorTransform = new ColorTransform(1, 1, 1, 1, 255, 0, 0, 0);
trace(colorTrans_1);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=255,
greenOffset=0, blueOffset=0, alphaOffset=0)

var colorTrans_2:ColorTransform = new ColorTransform(1, 1, 1, .5, 0, 0, 0, 0);
trace(colorTrans_2);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=0.5, redOffset=0,
greenOffset=0, blueOffset=0, alphaOffset=0)

colorTrans_1.concat(colorTrans_2);
trace(colorTrans_1);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=0.5, redOffset=255,
greenOffset=0, blueOffset=0, alphaOffset=0)

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans_1;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

greenMultiplier (ColorTransform.greenMultiplier property)

```
public greenMultiplier : Number
```

A decimal value that is multiplied by the green channel value.

Availability

Flash Lite 3.1

Example

The following example creates the ColorTransform object `colorTrans` and adjusts its `greenMultiplier` from 1 to .5.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.greenMultiplier); // 1

colorTrans.greenMultiplier = .5;
trace(colorTrans.greenMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0x00FF00), this;
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

greenOffset (ColorTransform.greenOffset property)

```
public greenOffset : Number
```

A number from -255 to 255 that is added to the green channel value after it has been multiplied by the `greenMultiplier` value.

Availability

Flash Lite 2.0

Example

The following example creates the ColorTransform object `colorTrans` and adjusts its `greenOffset` value from 0 to 255.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.greenOffset); // 0

colorTrans.greenOffset = 255;
trace(colorTrans.greenOffset); // 255

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

redMultiplier (ColorTransform.redMultiplier property)

```
public redMultiplier : Number
```

A decimal value that is multiplied by the red channel value.

Availability

Flash Lite 2.0

Example

The following example creates the ColorTransform object `colorTrans` and adjusts its `redMultiplier` value from 1 to .5.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.redMultiplier); // 1

colorTrans.redMultiplier = .5;
trace(colorTrans.redMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

redOffset (ColorTransform.redOffset property)

```
public redOffset : Number
```

A number from -255 to 255 that is added to the red channel value after it has been multiplied by the `redMultiplier` value.

Availability

Flash Lite 2.0

Example

The following example creates the ColorTransform object `colorTrans` and adjusts its `redOffset` value from 0 to 255.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.redOffset); // 0

colorTrans.redOffset = 255;
trace(colorTrans.redOffset); // 255

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

rgb (ColorTransform.rgb property)

```
public rgb : Number
```

The RGB color value for a ColorTransform object.

When you set this property, it changes the three color offset values (`redOffset`, `greenOffset`, and `blueOffset`), and sets the three color multiplier values (`redMultiplier`, `greenMultiplier`, and `blueMultiplier`) to 0. The alpha transparency multiplier and offset values do not change.

Pass a value for this property in the format: 0x*RRGGBB*. *RR*, *GG*, and *BB* each consist of two hexadecimal digits that specify the offset of each color component. The 0x tells the ActionScript compiler that the number is a hexadecimal value.

Availability

Flash Lite 2.0

Example

The following example creates the ColorTransform object `colorTrans` and adjusts its `rgb` value to 0xFF0000.

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.rgb); // 0

colorTrans.rgb = 0xFF0000;
trace(colorTrans.rgb); // 16711680
trace("0x" + colorTrans.rgb.toString(16)); // 0xff0000

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

toString (ColorTransform.toString method)

```
public toString() : String
```

Formats and returns a string that describes all of the properties of the ColorTransform object.

Availability

Flash Lite 2.0

Returns

String A string that lists all of the properties of the ColorTransform object.

Example

The following example creates the ColorTransform object `colorTrans` and calls its `toSting()` method. This method results in a string with the following format: (redMultiplier=RM, greenMultiplier=GM, blueMultiplier=BM, alphaMultiplier=AM, redOffset=RO, greenOffset=GO, blueOffset=BO, alphaOffset=AO).

```
import flash.geom.ColorTransform;

var colorTrans:ColorTransform = new ColorTransform(1, 2, 3, 4, -255, -128, 128, 255);
trace(colorTrans.toString());
// (redMultiplier=1, greenMultiplier=2, blueMultiplier=3, alphaMultiplier=4, redOffset=-255,
greenOffset=-128, blueOffset=128, alphaOffset=255)
```

Date

```
Object
    |
    +-Date
```

```
public class Date
extends Object
```

The Date class lets you retrieve date and time values relative to Universal Time (Greenwich Mean Time, now called universal time or UTC) or relative to the operating system on which Flash Lite player is running. The methods of the Date class are not static but apply only to the individual Date object specified when the method is called. The `Date.UTC()` method is an exception; it is a static method.

The Date class handles daylight saving time differently, depending on the operating system and Flash Player version. Flash Player 6 and later versions handle daylight saving time on the following operating systems in these ways:

- Windows - the Date object automatically adjusts its output for daylight saving time. The Date object detects whether daylight saving time is employed in the current locale, and if so, it detects the standard-to-daylight saving time transition date and times. However, the transition dates currently in effect are applied to dates in the past and the future, so the daylight saving time bias might calculate incorrectly for dates in the past when the locale had different transition dates.
- Mac OS X - the Date object automatically adjusts its output for daylight saving time. The time zone information database in Mac OS X is used to determine whether any date or time in the present or past should have a daylight saving time bias applied.
- Mac OS 9 - the operating system provides only enough information to determine whether the current date and time should have a daylight saving time bias applied. Accordingly, the date object assumes that the current daylight saving time bias applies to all dates and times in the past or future.

Flash Player 5 handles daylight saving time on the following operating systems as follows:

- Windows - the U.S. rules for daylight saving time are always applied, which leads to incorrect transitions in Europe and other areas that employ daylight saving time but have different transition times than the U.S. Flash correctly detects whether daylight saving time is used in the current locale.

  To call the methods of the Date class, you must first create a Date object using the constructor for the Date class, described later in this section.

Availability

Flash Lite 2.0

Property summary

Properties inherited from class Object

```
constructor (Object.constructor property),__proto__ (Object.__proto__
property)prototype (Object.prototype property),__resolve (Object.__resolve property)
```

Constructor summary

Signature	Description
Date ([yearOrTimevalue:Number], [month:Number], [date:Number], [hour:Number], [minute:Number], [second:Number], [millisecond:Number])	Constructs a new Date object that holds the specified date and time.

Method summary

Modifiers	Signature	Description
	getDate () : Number	Returns the day of the month (an integer from 1 to 31) of the specified Date object according to local time.
	getDay () : Number	Returns the day of the week (0 for Sunday, 1 for Monday, and so on) of the specified Date object according to local time.
	getFullYear () : Number	Returns the full year (a four-digit number, such as 2000) of the specified Date object, according to local time.
	getHours () : Number	Returns the hour (an integer from 0 to 23) of the specified Date object, according to local time.
	getLocaleLongDate () : String	Returns a string representing the current date, in long form, formatted according to the currently defined locale.
	getLocaleShortDate () : String	Returns a string representing the current date, in short form, formatted according to the currently defined locale.
	getLocaleTime () : String	Returns a string representing the current time, formatted according to the currently defined locale.
	getMilliseconds () : Number	Returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to local time.
	getMinutes () : Number	Returns the minutes (an integer from 0 to 59) of the specified Date object, according to local time.
	getMonth () : Number	Returns the month (0 for January, 1 for February, and so on) of the specified Date object, according to local time.
	getSeconds () : Number	Returns the seconds (an integer from 0 to 59) of the specified Date object, according to local time.
	getTime () : Number	Returns the number of milliseconds since midnight January 1, 1970, universal time, for the specified Date object.
	getTimezoneOffset () : Number	Returns the difference, in minutes, between the computer's local time and universal time.
	getUTCDate () : Number	Returns the day of the month (an integer from 1 to 31) in the specified Date object, according to universal time.
	getUTCDay () : Number	Returns the day of the week (0 for Sunday, 1 for Monday, and so on) of the specified Date object, according to universal time.
	getUTCFullYear () : Number	Returns the four-digit year of the specified Date object, according to universal time.
	getUTCHours () : Number	Returns the hour (an integer from 0 to 23) of the specified Date object, according to universal time.

Modifiers	Signature	Description
	getUTCMilliseconds() : Number	Returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to universal time.
	getUTCMinutes() : Number	Returns the minutes (an integer from 0 to 59) of the specified Date object, according to universal time.
	getUTCMonth() : Number	Returns the month (0 [January] to 11 [December]) of the specified Date object, according to universal time.
	getUTCSeconds() : Number	Returns the seconds (an integer from 0 to 59) of the specified Date object, according to universal time.
	getUTCYear() : Number	Returns the year of this `Date` according to universal time (UTC).
	getYear() : Number	Returns the year of the specified Date object, according to local time.
	setDate(date:Number) : Number	Sets the day of the month for the specified Date object, according to local time, and returns the new time in milliseconds.
	setFullYear(year:Number, [month:Number], [date:Number]) : Number	Sets the year of the specified Date object, according to local time and returns the new time in milliseconds.
	setHours(hour:Number) : Number	Sets the hours for the specified Date object according to local time and returns the new time in milliseconds.
	setMilliseconds(millisecond:Number) : Number	Sets the milliseconds for the specified Date object according to local time and returns the new time in milliseconds.
	setMinutes(minute:Number) : Number	Sets the minutes for a specified Date object according to local time and returns the new time in milliseconds.
	setMonth(month:Number, [date:Number]) : Number	Sets the month for the specified Date object in local time and returns the new time in milliseconds.
	setSeconds(second:Number) : Number	Sets the seconds for the specified Date object in local time and returns the new time in milliseconds.
	setTime(millisecond:Number) : Number	Sets the date for the specified Date object in milliseconds since midnight on January 1, 1970, and returns the new time in milliseconds.
	setUTCDate(date:Number) : Number	Sets the date for the specified Date object in universal time and returns the new time in milliseconds.
	setUTCFullYear(year:Number, [month:Number], [date:Number]) : Number	Sets the year for the specified Date object (*my_date*) in universal time and returns the new time in milliseconds.
	setUTCHours(hour:Number, [minute:Number], [second:Number], [millisecond:Number]) : Number	Sets the hour for the specified Date object in universal time and returns the new time in milliseconds.

Modifiers	Signature	Description
	setUTCMilliseconds(millisecond:Number) : Number	Sets the milliseconds for the specified Date object in universal time and returns the new time in milliseconds.
	setUTCMinutes(minute:Number, [second:Number], [millisecond:Number]) : Number	Sets the minute for the specified Date object in universal time and returns the new time in milliseconds.
	setUTCMonth(month:Number, [date:Number]) : Number	Sets the month, and optionally the day, for the specified Date object in universal time and returns the new time in milliseconds.
	setUTCSeconds(second:Number, [millisecond:Number]) : Number	Sets the seconds for the specified Date object in universal time and returns the new time in milliseconds.
	setYear(year:Number) : Number	Sets the year for the specified Date object in local time and returns the new time in milliseconds.
	toString() : String	Returns a string value for the specified date object in a readable format.
static	UTC(year:Number, month:Number, [date:Number], [hour:Number], [minute:Number], [second:Number], [millisecond:Number]) : Number	Returns the number of milliseconds between midnight on January 1, 1970, universal time, and the time specified in the parameters.
	valueOf() : Number	Returns the number of milliseconds since midnight January 1, 1970, universal time, for this Date.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

Date constructor

```
public Date([yearOrTimevalue:Number], [month:Number], [date:Number], [hour:Number],
[minute:Number], [second:Number], [millisecond:Number])
```

Constructs a new Date object that holds the specified date and time.

The `Date()` constructor takes up to seven parameters to specify a date and time to the millisecond. Alternatively, you can pass a single value to the `Date()` constructor that indicates a time value based on the number of milliseconds since January 1, 1970 0:00:000 GMT. Or you can specify no parameters, and the `Date()` date object is assigned the current date and time.

The following code shows several different ways to create a Date object:

```
var d1:Date = new Date();
var d3:Date = new Date(2000, 0, 1);
var d4:Date = new Date(65, 2, 6, 9, 30, 15, 0);
var d5:Date = new Date(-14159025000);
```

In the first line of code, a Date object is set to the time when the assignment statement is run.

In the second line, a Date object is created with year, month, and date parameters passed to it, resulting in the time 0:00:00 GMT January 1, 2000.

In the third line, a Date object is created with year, month, and date parameters passed to it, resulting in the time 09:30:15 GMT (+ 0 milliseconds) March 6, 1965. Note that since the year parameter is specified as a two-digit integer, it is interpreted as 1965.

In the fourth line, only one parameter is passed, which is a time value representing the number of milliseconds before or after 0:00:00 GMT January 1, 1970; since the value is negative, it represents a time *before* 0:00:00 GMT January 1, 1970, and in this case the time is 02:56:15 GMT July, 21 1969.

Availability

Flash Lite 2.0

Parameters

yearOrTimevalue : Number [optional] - If other parameters are specified, this number represents a year (such as 1965); otherwise, it represents a time value. If the number represents a year, a value of 0 to 99 indicates 1900 through 1999; otherwise all four digits of the year must be specified. If the number represents a time value (no other parameters are specified), it is the number of milliseconds before or after 0:00:00 GMT January 1, 1970; a negative value represents a time *before* 0:00:00 GMT January 1, 1970, and a positive value represents a time after.

month : Number [optional] - An integer from 0 (January) to 11 (December).

date : Number [optional] - An integer from 1 to 31.

hour : Number [optional] - An integer from 0 (midnight) to 23 (11 p.m.).

minute : Number [optional] - An integer from 0 to 59.

second : Number [optional] - An integer from 0 to 59.

millisecond : Number [optional] - An integer from 0 to 999 of milliseconds.

Example

The following example retrieves the current date and time:

```
var now_date:Date = new Date();
```

The following example creates a new Date object for Mary's birthday, August 12, 1974 (because the month parameter is zero-based, the example uses 7 for the month, not 8):

```
var maryBirthday:Date = new Date (74, 7, 12);
```

The following example creates a new Date object and concatenates the returned values of `Date.getMonth()`, `Date.getDate()`, and `Date.getFullYear()`:

```
var today_date:Date = new Date();
var date_str:String =
((today_date.getMonth()+1)+"/"+today_date.getDate()+"/"+today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

See also

getMonth (Date.getMonth method), getDate (Date.getDate method), getFullYear (Date.getFullYear method)

getDate (Date.getDate method)

```
public getDate() : Number
```

Returns the day of the month (an integer from 1 to 31) of the specified Date object according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a new Date object and concatenates the returned values of `Date.getMonth()`, `Date.getDate()`, and `Date.getFullYear()`:

```
var today_date:Date = new Date();
var date_str:String =
(today_date.getDate()+"/"+(today_date.getMonth()+1)+"/"+today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

See also

getMonth (Date.getMonth method), getFullYear (Date.getFullYear method)

getDay (Date.getDay method)

```
public getDay() : Number
```

Returns the day of the week (0 for Sunday, 1 for Monday, and so on) of the specified Date object according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Returns

Number -- An integer representing the day of the week.

Example

The following example creates a new Date object and uses `getDay()` to determine the current day of the week:

```
var dayOfWeek_array:Array = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday");
var today_date:Date = new Date();
var day_str:String = dayOfWeek_array[today_date.getDay()];
trace("Today is "+day_str);
```

getFullYear (Date.getFullYear method)

```
public getFullYear() : Number
```

Returns the full year (a four-digit number, such as 2000) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Returns

Number - An integer representing the year.

Example

The following example uses the constructor to create a Date object. The trace statement shows the value returned by the `getFullYear()` method.

```
var my_date:Date = new Date();
trace(my_date.getYear()); // displays 104
trace(my_date.getFullYear()); // displays current year
```

getHours (Date.getHours method)

```
public getHours() : Number
```

Returns the hour (an integer from 0 to 23) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example uses the constructor to create a Date object based on the current time and uses the `getHours()` method to display hour values from that object:

```
var my_date:Date = new Date();
trace(my_date.getHours());

var my_date:Date = new Date();
var hourObj:Object = getHoursAmPm(my_date.getHours());
trace(hourObj.hours);
trace(hourObj.ampm);

function getHoursAmPm(hour24:Number):Object {
    var returnObj:Object = new Object();
    returnObj.ampm = (hour24<12) ? "AM" : "PM";
    var hour12:Number = hour24%12;
    if (hour12 == 0) {
    hour12 = 12;
    }
    returnObj.hours = hour12;
    return returnObj;
}
```

getLocaleLongDate (Date.getLocaleLongDate method)

```
public getLocaleLongDate() : String
```

Returns a string representing the current date, in long form, formatted according to the currently defined locale.

Note: The format of the date depends on the mobile device and the locale.

Availability
Flash Lite 2.0

Returns
String - A string representing the current date, in long form, formatted according to the currently defined locale.

Example
The following example uses the constructor to create a Date object based on the current time. It also uses the `getLocaleLongDate()` method to return the current date, in long form, formatted according to the currently defined locale, as follows:

```
var my_date:Date = new Date();
trace(my_date.getLocaleLongDate());
```

The following are sample return values that `getLocaleLongDate()` returns:

```
October 16, 2005
16 October 2005
```

getLocaleShortDate (Date.getLocaleShortDate method)

```
public getLocaleShortDate() : String
```

Returns a string representing the current date, in short form, formatted according to the currently defined locale.

Note: The format of the date depends on the mobile device and the locale.

Availability
Flash Lite 2.0

Returns
String - A string representing the current date, in short form, formatted according to the currently defined locale.

Example
The following example uses the constructor to create a Date object based on the current time. It also uses the `getLocaleShortDate()` method to return the current date, in short form, formatted according to the currently defined locale, as follows:

```
var my_date:Date = new Date();
trace(my_date.getLocaleShortDate());
```

The following are sample return values that `getLocaleLongDate()` returns:

```
10/16/2005
16-10-2005
```

getLocaleTime (Date.getLocaleTime method)

```
public getLocaleTime() : String
```

Returns a string representing the current time, formatted according to the currently defined locale.

Note: The format of the date depends on the mobile device and the locale.

Availability
Flash Lite 2.0

Returns
String - A string representing the current time, formatted according to the currently defined locale.

Example
The following example uses the constructor to create a Date object based on the current time. It also uses the `getLocaleTime()` method to return the time of the current locale, as follows:

```
var my_date:Date = new Date();
trace(my_date.getLocaleTime());
```

The following are sample return values that `getLocaleTime()` returns:

```
6:10:44 PM
18:10:44
```

getMilliseconds (Date.getMilliseconds method)

```
public getMilliseconds() : Number
```

Returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability
Flash Lite 2.0

Returns
Number An integer.

Example
The following example uses the constructor to create a Date object based on the current time and uses the `getMilliseconds()` method to return the milliseconds value from that object:

```
var my_date:Date = new Date();
trace(my_date.getMilliseconds());
```

getMinutes (Date.getMinutes method)

```
public getMinutes() : Number
```

Returns the minutes (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example uses the constructor to create a Date object based on the current time, and uses the `getMinutes()` method to return the minutes value from that object:

```
var my_date:Date = new Date();
trace(my_date.getMinutes());
```

getMonth (Date.getMonth method)

```
public getMonth() : Number
```

Returns the month (0 for January, 1 for February, and so on) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example uses the constructor to create a Date object based on the current time and uses the `getMonth()` method to return the month value from that object:

```
var my_date:Date = new Date();
trace(my_date.getMonth());
```

The following example uses the constructor to create a Date object based on the current time and uses the `getMonth()` method to display the current month as a numeric value, and display the name of the month.

```
var my_date:Date = new Date();
trace(my_date.getMonth());
trace(getMonthAsString(my_date.getMonth()));
function getMonthAsString(month:Number):String {
    var monthNames_array:Array = new Array("January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November", "December");
    return monthNames_array[month];
}
```

getSeconds (Date.getSeconds method)

```
public getSeconds() : Number
```

Returns the seconds (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Lite player is running.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example uses the constructor to create a Date object based on the current time and uses the `getSeconds()` method to return the seconds value from that object:

```
var my_date:Date = new Date();
trace(my_date.getSeconds());
```

getTime (Date.getTime method)

```
public getTime() : Number
```

Returns the number of milliseconds since midnight January 1, 1970, universal time, for the specified Date object. Use this method to represent a specific instant in time when comparing two or more Date objects.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example uses the constructor to create a Date object based on the current time, and uses the `getTime()` method to return the number of milliseconds since midnight January 1, 1970:

```
var my_date:Date = new Date();
trace(my_date.getTime());
```

getTimezoneOffset (Date.getTimezoneOffset method)

```
public getTimezoneOffset() : Number
```

Returns the difference, in minutes, between the computer's local time and universal time.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example returns the difference between the local daylight saving time for San Francisco and universal time. Daylight saving time is factored into the returned result only if the date defined in the Date object occurs during daylight saving time. The output in this example is 420 minutes and displays in the Output panel (7 hours * 60 minutes/hour = 420 minutes). This example is Pacific Daylight Time (PDT, GMT-0700). The result varies depending on location and time of year.

```
var my_date:Date = new Date();
trace(my_date.getTimezoneOffset());
```

getUTCDate (Date.getUTCDate method)

```
public getUTCDate() : Number
```

Returns the day of the month (an integer from 1 to 31) in the specified Date object, according to universal time.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example creates a new Date object and uses `Date.getUTCDate()` and `Date.getDate()`. The value returned by `Date.getUTCDate()` can differ from the value returned by `Date.getDate()`, depending on the relationship between your local time zone and universal time.

```
var my_date:Date = new Date(2004,8,25);
trace(my_date.getUTCDate()); // output: 25
```

See also
`getDate (Date.getDate method)`

getUTCDay (Date.getUTCDay method)

```
public getUTCDay() : Number
```

Returns the day of the week (0 for Sunday, 1 for Monday, and so on) of the specified Date object, according to universal time.

Availability
Flash Lite 2.0

Returns

Number An integer.

Example

The following example creates a new Date object and uses `Date.getUTCDay()` and `Date.getDay()`. The value returned by `Date.getUTCDay()` can differ from the value returned by `Date.getDay()`, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getDay()); // output will be based on local timezone
trace(today_date.getUTCDay()); // output will equal getDay() plus or minus one
```

See also

`getDay (Date.getDay method)`

getUTCFullYear (Date.getUTCFullYear method)

```
public getUTCFullYear() : Number
```

Returns the four-digit year of the specified Date object, according to universal time.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a new Date object and uses `Date.getUTCFullYear()` and `Date.getFullYear()`. The value returned by `Date.getUTCFullYear()` may differ from the value returned by `Date.getFullYear()` if today's date is December 31 or January 1, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getFullYear()); // display based on local timezone
trace(today_date.getUTCFullYear()); // displays getYear() plus or minus 1
```

See also

`getFullYear (Date.getFullYear method)`

getUTCHours (Date.getUTCHours method)

```
public getUTCHours() : Number
```

Returns the hour (an integer from 0 to 23) of the specified Date object, according to universal time.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a new Date object and uses `Date.getUTCHours()` and `Date.getHours()`. The value returned by `Date.getUTCHours()` may differ from the value returned by `Date.getHours()`, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getHours()); // display based on local timezone
trace(today_date.getUTCHours()); // display equals getHours() plus or minus 12
```

See also

`getHours (Date.getHours method)`

getUTCMilliseconds (Date.getUTCMilliseconds method)

```
public getUTCMilliseconds() : Number
```

Returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to universal time.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a new Date object and uses `getUTCMilliseconds()` to return the milliseconds value from the Date object.

```
var today_date:Date = new Date();
trace(today_date.getUTCMilliseconds());
```

getUTCMinutes (Date.getUTCMinutes method)

```
public getUTCMinutes() : Number
```

Returns the minutes (an integer from 0 to 59) of the specified Date object, according to universal time.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a new Date object and uses `getUTCMinutes()` to return the minutes value from the Date object:

```
var today_date:Date = new Date();
trace(today_date.getUTCMinutes());
```

getUTCMonth (Date.getUTCMonth method)

```
public getUTCMonth() : Number
```

Returns the month (0 [January] to 11 [December]) of the specified Date object, according to universal time.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a new Date object and uses `Date.getUTCMonth()` and `Date.getMonth()`. The value returned by `Date.getUTCMonth()` can differ from the value returned by `Date.getMonth()` if today's date is the first or last day of a month, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getMonth()); // output based on local timezone
trace(today_date.getUTCMonth()); // output equals getMonth() plus or minus 1
```

See also

`getMonth (Date.getMonth method)`

getUTCSeconds (Date.getUTCSeconds method)

```
public getUTCSeconds() : Number
```

Returns the seconds (an integer from 0 to 59) of the specified Date object, according to universal time.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a new Date object and uses `getUTCSeconds()` to return the seconds value from the Date object:

```
var today_date:Date = new Date();
trace(today_date.getUTCSeconds());
```

getUTCYear (Date.getUTCYear method)

```
public getUTCYear() : Number
```

Returns the year of this `Date` according to universal time (UTC). The year is the full year minus 1900. For example, the year 2000 is represented as 100.

Availability

Flash Lite 2.0

Returns
Number -

Example
The following example creates a new Date object and uses `Date.getUTCFullYear()` and `Date.getFullYear()`. The value returned by `Date.getUTCFullYear()` may differ from the value returned by `Date.getFullYear()` if today's date is December 31 or January 1, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();

trace(today_date.getFullYear()); // display based on local timezone

trace(today_date.getUTCFullYear()); // displays getYear() plus or minus 1
```

getYear (Date.getYear method)

```
public getYear() : Number
```

Returns the year of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Lite player is running. The year is the full year minus 1900. For example, the year 2000 is represented as 100.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example creates a Date object with the month and year set to May 2004. The `Date.getYear()` method returns 104, and `Date.getFullYear()` returns 2004:

```
var today_date:Date = new Date(2004,4);
trace(today_date.getYear()); // output: 104
trace(today_date.getFullYear()); // output: 2004
```

See also
`getFullYear (Date.getFullYear method)`

setDate (Date.setDate method)

```
public setDate(date:Number) : Number
```

Sets the day of the month for the specified Date object, according to local time, and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Lite player is running.

Availability
Flash Lite 2.0

Parameters
date : Number - An integer from 1 to 31.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setDate()` to change the date to May 25, 2004:

```
var today_date:Date = new Date(2004,4,15);
trace(today_date.getDate()); //displays 15
today_date.setDate(25);
trace(today_date.getDate()); //displays 25
```

setFullYear (Date.setFullYear method)

```
public setFullYear(year:Number, [month:Number], [date:Number]) : Number
```

Sets the year of the specified Date object, according to local time and returns the new time in milliseconds. If the `month` and `date` parameters are specified, they are set to local time. Local time is determined by the operating system on which Flash Lite player is running.

Calling this method does not modify the other fields of the specified Date object but `Date.getUTCDay()` and `Date.getDay()` can report a new value if the day of the week changes as a result of calling this method.

Availability

Flash Lite 2.0

Parameters

year : Number - A four-digit number specifying a year. Two-digit numbers do not represent four-digit years; for example, 99 is not the year 1999, but the year 99.

month : Number [optional] - An integer from 0 (January) to 11 (December). If you omit this parameter, the month field of the specified Date object will not be modified.

date : Number [optional] - A number from 1 to 31. If you omit this parameter, the date field of the specified Date object will not be modified.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setFullYear()` to change the date to May 15, 2002:

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getFullYear()); //output: 2004
my_date.setFullYear(2002);
trace(my_date.getFullYear()); //output: 2002
```

See also

`getUTCDay (Date.getUTCDay method)`, `getDay (Date.getDay method)`

setHours (Date.setHours method)

```
public setHours(hour:Number) : Number
```

Sets the hours for the specified Date object according to local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Parameters

hour : Number - An integer from 0 (midnight) to 23 (11 p.m.).

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and uses `Date.setHours()` to change the time to 4:00 p.m.:

```
var my_date:Date = new Date(2004,4,15,8);
trace(my_date.getHours()); // output: 8
my_date.setHours(16);
trace(my_date.getHours()); // output: 16
```

setMilliseconds (Date.setMilliseconds method)

```
public setMilliseconds(millisecond:Number) : Number
```

Sets the milliseconds for the specified Date object according to local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Parameters

millisecond : Number - An integer from 0 to 999.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the date to 8:30 a.m. on May 15, 2004 with the milliseconds value set to 250, and then uses `Date.setMilliseconds()` to change the milliseconds value to 575:

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getMilliseconds()); // output: 250
my_date.setMilliseconds(575);
trace(my_date.getMilliseconds()); // output: 575
```

setMinutes (Date.setMinutes method)

```
public setMinutes(minute:Number) : Number
```

Sets the minutes for a specified Date object according to local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Parameters

minute : Number - An integer from 0 to 59.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and then uses `Date.setMinutes()` to change the time to 8:30 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getMinutes()); // output: 0
my_date.setMinutes(30);
trace(my_date.getMinutes()); // output: 30
```

setMonth (Date.setMonth method)

```
public setMonth(month:Number, [date:Number]) : Number
```

Sets the month for the specified Date object in local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Parameters

month : Number - An integer from 0 (January) to 11 (December).

date : Number [optional] - An integer from 1 to 31. If you omit this parameter, the date field of the specified Date object will not be modified.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setMonth()` to change the date to June 15, 2004:

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getMonth()); //output: 4
my_date.setMonth(5);
trace(my_date.getMonth()); //output: 5
```

setSeconds (Date.setSeconds method)

```
public setSeconds(second:Number) : Number
```

Sets the seconds for the specified Date object in local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Lite player is running.

Availability

Flash Lite 2.0

Parameters

second : Number - An integer from 0 to 59.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the time and date to 8:00:00 a.m. on May 15, 2004, and uses `Date.setSeconds()` to change the time to 8:00:45 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getSeconds()); // output: 0
my_date.setSeconds(45);
trace(my_date.getSeconds()); // output: 45
```

setTime (Date.setTime method)

```
public setTime(millisecond:Number) : Number
```

Sets the date for the specified Date object in milliseconds since midnight on January 1, 1970, and returns the new time in milliseconds.

Availability

Flash Lite 2.0

Parameters

millisecond : Number - A number; an integer value where 0 is midnight on January 1, universal time.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and uses `Date.setTime()` to change the time to 8:30 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
var myDate_num:Number = my_date.getTime(); // convert my_date to milliseconds
myDate_num += 30 * 60 * 1000; // add 30 minutes in milliseconds
my_date.setTime(myDate_num); // set my_date Date object 30 minutes forward
trace(my_date.getFullYear()); // output: 2004
trace(my_date.getMonth()); // output: 4
trace(my_date.getDate()); // output: 15
trace(my_date.getHours()); // output: 8
trace(my_date.getMinutes()); // output: 30
```

setUTCDate (Date.setUTCDate method)

```
public setUTCDate(date:Number) : Number
```

Sets the date for the specified Date object in universal time and returns the new time in milliseconds. Calling this method does not modify the other fields of the specified Date object, but `Date.getUTCDay()` and `Date.getDay()` can report a new value if the day of the week changes as a result of calling this method.

Availability

Flash Lite 2.0

Parameters

date : Number - A number; an integer from 1 to 31.

Returns

Number - An integer.

Example

The following example initially creates a new Date object with today's date, uses `Date.setUTCDate()` to change the date value to 10, and changes it again to 25:

```
var my_date:Date = new Date();
my_date.setUTCDate(10);
trace(my_date.getUTCDate()); // output: 10
my_date.setUTCDate(25);
trace(my_date.getUTCDate()); // output: 25
```

See also

getUTCDay (Date.getUTCDay method), getDay (Date.getDay method)

setUTCFullYear (Date.setUTCFullYear method)

```
public setUTCFullYear(year:Number, [month:Number], [date:Number]) : Number
```

Sets the year for the specified Date object (*my_date*) in universal time and returns the new time in milliseconds.

Optionally, this method can also set the month and date represented by the specified Date object. Calling this method does not modify the other fields of the specified Date object, but `Date.getUTCDay()` and `Date.getDay()` can report a new value if the day of the week changes as a result of calling this method.

Availability

Flash Lite 2.0

Parameters

year : Number - An integer that represents the year specified as a full four-digit year, such as 2000.

month : Number [optional] - An integer from 0 (January) to 11 (December). If you omit this parameter, the month field of the specified Date object will not be modified.

date : Number [optional] - An integer from 1 to 31. If you omit this parameter, the date field of the specified Date object will not be modified.

Returns

Number - An integer.

Example

The following example initially creates a new Date object with today's date, uses `Date.setUTCFullYear()` to change the year value to 2001, and changes the date to May 25, 1995:

```
var my_date:Date = new Date();
my_date.setUTCFullYear(2001);
trace(my_date.getUTCFullYear()); // output: 2001
my_date.setUTCFullYear(1995, 4, 25);
trace(my_date.getUTCFullYear()); // output: 1995
trace(my_date.getUTCMonth()); // output: 4
trace(my_date.getUTCDate()); // output: 25
```

See also

`getUTCDay (Date.getUTCDay method)`, `getDay (Date.getDay method)`

setUTCHours (Date.setUTCHours method)

```
public setUTCHours(hour:Number, [minute:Number], [second:Number], [millisecond:Number]) :
Number
```

Sets the hour for the specified Date object in universal time and returns the new time in milliseconds.

Availability

Flash Lite 2.0

Parameters

hour : Number - A number; an integer from 0 (midnight) to 23 (11 p.m.).

minute : Number [optional] - A number; an integer from 0 to 59. If you omit this parameter, the minutes field of the specified Date object will not be modified.

second : Number [optional] - A number; an integer from 0 to 59. If you omit this parameter, the seconds field of the specified Date object will not be modified.

millisecond : Number [optional] - A number; an integer from 0 to 999. If you omit this parameter, the milliseconds field of the specified Date object will not be modified.

Returns

Number - An integer.

Example
The following example initially creates a new Date object with today's date, uses `Date.setUTCHours()` to change the time to 8:30 a.m., and changes the time again to 5:30:47 p.m.:

```
var my_date:Date = new Date();
my_date.setUTCHours(8,30);
trace(my_date.getUTCHours()); // output: 8
trace(my_date.getUTCMinutes()); // output: 30
my_date.setUTCHours(17,30,47);
trace(my_date.getUTCHours()); // output: 17
trace(my_date.getUTCMinutes()); // output: 30
trace(my_date.getUTCSeconds()); // output: 47
```

setUTCMilliseconds (Date.setUTCMilliseconds method)

```
public setUTCMilliseconds(millisecond:Number) : Number
```

Sets the milliseconds for the specified Date object in universal time and returns the new time in milliseconds.

Availability
Flash Lite 2.0

Parameters
millisecond : Number - An integer from 0 to 999.

Returns
Number - An integer.

Example
The following example initially creates a new Date object, setting the date to 8:30 a.m. on May 15, 2004 with the milliseconds value set to 250, and uses `Date.setUTCMilliseconds()` to change the milliseconds value to 575:

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getUTCMilliseconds()); // output: 250
my_date.setUTCMilliseconds(575);
trace(my_date.getUTCMilliseconds()); // output: 575
```

setUTCMinutes (Date.setUTCMinutes method)

```
public setUTCMinutes(minute:Number, [second:Number], [millisecond:Number]) : Number
```

Sets the minute for the specified Date object in universal time and returns the new time in milliseconds.

Availability
Flash Lite 2.0

Parameters
minute : Number - An integer from 0 to 59.

second : Number [optional] - An integer from 0 to 59. If you omit this parameter, the seconds field of the specified Date object will not be modified.

millisecond : Number [optional] - An integer from 0 to 999. If you omit this parameter, the milliseconds field of the specified Date object will not be modified.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and uses `Date.setUTCMinutes()` to change the time to 8:30 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getUTCMinutes()); // output: 0
my_date.setUTCMinutes(30);
trace(my_date.getUTCMinutes()); // output: 30
```

setUTCMonth (Date.setUTCMonth method)

```
public setUTCMonth(month:Number, [date:Number]) : Number
```

Sets the month, and optionally the day, for the specified Date object in universal time and returns the new time in milliseconds. Calling this method does not modify the other fields of the specified Date object, but `Date.getUTCDay()` and `Date.getDay()` might report a new value if the day of the week changes as a result of specifying a value for the `date` parameter.

Availability

Flash Lite 2.0

Parameters

month : Number - An integer from 0 (January) to 11 (December).

date : Number [optional] - An integer from 1 to 31. If you omit this parameter, the date field of the specified Date object will not be modified.

Returns

Number - An integer.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setMonth()` to change the date to June 15, 2004:

```
var today_date:Date = new Date(2004,4,15);
trace(today_date.getUTCMonth()); // output: 4
today_date.setUTCMonth(5);
trace(today_date.getUTCMonth()); // output: 5
```

See also

`getUTCDay (Date.getUTCDay method)`, `getDay (Date.getDay method)`

setUTCSeconds (Date.setUTCSeconds method)

```
public setUTCSeconds(second:Number, [millisecond:Number]) : Number
```

Sets the seconds for the specified Date object in universal time and returns the new time in milliseconds.

Availability
Flash Lite 2.0

Parameters
second : Number - An integer from 0 to 59.

millisecond : Number [optional] - An integer from 0 to 999. If you omit this parameter, the milliseconds field of the specified Date object will not be modified.

Returns
Number - An integer.

Example
The following example initially creates a new Date object, setting the time and date to 8:00:00 a.m. on May 15, 2004, and uses `Date.setSeconds()` to change the time to 8:30:45 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getUTCSeconds()); // output: 0
my_date.setUTCSeconds(45);
trace(my_date.getUTCSeconds()); // output: 45
```

setYear (Date.setYear method)

```
public setYear(year:Number) : Number
```

Sets the year for the specified Date object in local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Lite player is running.

Availability
Flash Lite 2.0

Parameters
year : Number - A number that represents the year. If `year` is an integer between 0 and 99, `setYear` sets the year at 1900 + `year`; otherwise, the year is the value of the `year` parameter.

Returns
Number - An integer.

Example
The following example creates a new Date object with the date set to May 25, 2004, uses `setYear()` to change the year to 1999, and changes the year to 2003:

```
var my_date:Date = new Date(2004,4,25);
trace(my_date.getYear()); // output: 104
trace(my_date.getFullYear()); // output: 2004
my_date.setYear(99);
trace(my_date.getYear()); // output: 99
trace(my_date.getFullYear()); // output: 1999
my_date.setYear(2003);
trace(my_date.getYear()); // output: 103
trace(my_date.getFullYear()); // output: 2003
```

toString (Date.toString method)

```
public toString() : String
```

Returns a string value for the specified date object in a readable format.

Availability

Flash Lite 2.0

Returns

String - A string.

Example

The following example returns the information in the `dateOfBirth_date` Date object as a string. The output from the trace statements are in local time and vary accordingly. For Pacific Daylight Time the output is seven hours earlier than universal time: Mon Aug 12 18:15:00 GMT-0700 1974.

```
var dateOfBirth_date:Date = new Date(74, 7, 12, 18, 15);
trace (dateOfBirth_date);
trace (dateOfBirth_date.toString());
```

UTC (Date.UTC method)

```
public static UTC(year:Number, month:Number, [date:Number], [hour:Number], [minute:Number],
[second:Number], [millisecond:Number]) : Number
```

Returns the number of milliseconds between midnight on January 1, 1970, universal time, and the time specified in the parameters. This is a static method that is invoked through the Date object constructor, not through a specific Date object. This method lets you create a Date object that assumes universal time, whereas the Date constructor assumes local time.

Availability

Flash Lite 2.0

Parameters

year : Number - A four-digit integer that represents the year (for example, 2000).

month : Number - An integer from 0 (January) to 11 (December).

date : Number [optional] - An integer from 1 to 31.

hour : Number [optional] - An integer from 0 (midnight) to 23 (11 p.m.).

minute : Number [optional] - An integer from 0 to 59.

second : Number [optional] - An integer from 0 to 59.

millisecond : Number [optional] - An integer from 0 to 999.

Returns
Number - An integer.

Example
The following example creates a new `maryBirthday_date` Date object defined in universal time. This is the universal time variation of the example used for the `new Date` constructor method. The output is in local time and varies accordingly. For Pacific Daylight Time the output is seven hours earlier than UTC: Sun Aug 11 17:00:00 GMT-0700 1974.

```
var maryBirthday_date:Date = new Date(Date.UTC(1974, 7, 12));
trace(maryBirthday_date);
```

valueOf (Date.valueOf method)

```
public valueOf() : Number
```

Returns the number of milliseconds since midnight January 1, 1970, universal time, for this `Date`.

Availability
Flash Lite 2.0

Returns
Number - The number of milliseconds.

Error

```
Object
    |
    +-Error
```

```
public class Error
extends Object
```

Contains information about an error that occurred in a script. You create an Error object using the `Error` constructor function. Typically, you `throw` a new Error object from within a `try` code block that is then caught by a `catch` or `finally` code block.

You can also create a subclass of the Error class and throw instances of that subclass.

Availability
Flash Lite 2.0

Property summary

Modifiers	Property	Description
	message : String	Contains the message associated with the Error object.
	name : String	Contains the name of the Error object.

Properties inherited from class Object

```
constructor (Object.constructor property), __proto__ (Object.__proto__
property) prototype (Object.prototype property), __resolve (Object.__resolve property)
```

Constructor summary

Signature	Description
Error ([message : String])	Creates a new Error object.

Method summary

Modifiers	Signature	Description
	toString () : String	Returns the string "Error" by default or the value contained in Error.message, if defined.

Methods inherited from class Object

```
addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty
method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf
(Object.isPrototypeOf method) registerClass (Object.registerClass method), toString
(Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf
method) watch (Object.watch method)
```

Error constructor

```
public Error([message:String])
```

Creates a new Error object. If you pass a *message* parameter, its value is assigned to the `Error.message` property.

Availability

Flash Lite 2.0

Parameters

message : String [optional] - A string associated with the Error object.

Example

In the following example, a function throws an error (with a specified message) if the two strings that are passed to it are not identical:

```
function compareStrings(str1_str:String, str2_str:String):Void {
    if (str1_str != str2_str) {
    throw new Error("Strings do not match.");
    }
}
try {
    compareStrings("Dog", "dog");
    // output: Strings do not match.
} catch (e_err:Error) {
    trace(e_err.toString());
}
```

See also

throw statement, try..catch..finally statement

message (Error.message property)

```
public message : String
```

The message associated with the Error object. By default, the value of this property is "Error". You can specify a message property when you create an Error object by passing the error string to the Error constructor function.

Availability

Flash Lite 2.0

Example

In the following example, a function throws a specified message depending on the parameters entered into theNum. If two numbers can be divided, SUCCESS and the number are shown. Specific errors are shown if you try to divide by 0 or enter only 1 parameter:

```
function divideNum(num1:Number, num2:Number):Number {
    if (isNaN(num1) || isNaN(num2)) {
    throw new Error("divideNum function requires two numeric parameters.");
    } else if (num2 == 0) {
    throw new Error("cannot divide by zero.");
    }
    return num1/num2;
}
try {
    var theNum:Number = divideNum(1, 0);
    trace("SUCCESS! "+theNum);
} catch (e_err:Error) {
    trace("ERROR! "+e_err.message);
    trace("\t"+e_err.name);
}
```

If you test this ActionScript without any modifications to the numbers you divide, you see an error displayed in the Output panel because you are trying to divide by 0.

See also

throw statement, try..catch..finally statement

name (Error.name property)

```
public name : String
```

Contains the name of the Error object. By default, the value of this property is "Error".

Availability

Flash Lite 2.0

Example

In the following example, a function throws a specified error depending on the two numbers that you try to divide. Add the following ActionScript to Frame 1 of the Timeline:

```
function divideNumber(numerator:Number, denominator:Number):Number {
    if (isNaN(numerator) || isNaN(denominator)) {
    throw new Error("divideNumber() function requires two numeric parameters.");
    } else if (denominator == 0) {
    throw new DivideByZeroError();
    }
    return numerator/denominator;
}
try {
    var theNum:Number = divideNumber(1, 0);
    trace("SUCCESS! "+theNum);
    // output: DivideByZeroError -> Unable to divide by zero.
} catch (e_err:DivideByZeroError) {
    // divide by zero error occurred
    trace(e_err.name+" -> "+e_err.toString());
} catch (e_err:Error) {
    // generic error occurred
    trace(e_err.name+" -> "+e_err.toString());
}
```

Add the following code to an .as file called DivideByZeroError.as and save the class file in the same directory as your .fla document.

```
class DivideByZeroError extends Error {
    var name:String = "DivideByZeroError";
    var message:String = "Unable to divide by zero.";
}
```

See also

throw statement, try..catch..finally statement

toString (Error.toString method)

```
public toString() : String
```

Returns the string "Error" or the value contained in Error.message, if defined.

Availability

Flash Lite 2.0

Returns

String A String

Example

In the following example, a function throws an error (with a specified message) if the two strings that are passed to it are not identical:

```
function compareStrings(str1_str:String, str2_str:String):Void {
if (str1_str != str2_str) {
    throw new Error("Strings do not match.");
    }
}
try {
    compareStrings("Dog", "dog");
    // output: Strings do not match.
} catch (e_err:Error) {
    trace(e_err.toString());
}
```

See also

message (Error.message property), throw statement, try..catch..finally statement

ExtendedKey

```
Object
    |
    +-ExtendedKey
```

```
public class ExtendedKey
extends Object
```

Provides extended key codes that can be returned from the `Key.getCode()` method.

Availability

Flash Lite 2.0

Example

The following example creates a listener that is called when a key is pressed. It uses the `Key.getCode()` method to get the key code for the key that was pressed:

```
var myListener = new Object();

myListener.onKeyDown = function() {
    var code = Key.getCode();
    trace(code + " down");
}

myListener.onKeyUp = function() {
    trace("onKeyUp called");
}

Key.addListener(myListener);
```

See also

getCode (Key.getCode method)

Property summary

Modifiers	Property	Description
static	SOFT1 : String	The key code value for the SOFT1 soft key.
static	SOFT3 : String	The key code value for the SOFT3 soft key.
static	SOFT4 : String	The key code value for the SOFT4 soft key.
static	SOFT5 : String	The key code value for the SOFT5 soft key.
static	SOFT6 : String	The key code value for the SOFT6 soft key.
static	SOFT7 : String	The key code value for the SOFT7 soft key.
static	SOFT8 : String	The key code value for the SOFT8 soft key.
static	SOFT9 : String	The key code value for the SOFT9 soft key.
static	SOFT10 : String	The key code value for the SOFT10 soft key.
static	SOFT11 : String	The key code value for the SOFT11 soft key.
static	SOFT12 : String	The key code value for the SOFT12 soft key.
static	SOFT2 : String	The key code value for the SOFT2 soft key.

Properties inherited from class Object

```
constructor (Object.constructor property),__proto__ (Object.__proto__
property)prototype (Object.prototype property),__resolve (Object.__resolve property)
```

Method summary

Methods inherited from class Object

```
addProperty (Object.addProperty method),hasOwnProperty (Object.hasOwnProperty
method)isPropertyEnumerable (Object.isPropertyEnumerable method)isPrototypeOf
(Object.isPrototypeOf method)registerClass (Object.registerClass method),toString
(Object.toString method)unwatch (Object.unwatch method),valueOf (Object.valueOf
method)watch (Object.watch method)
```

SOFT1 (ExtendedKey.SOFT1 property)

```
public static SOFT1 : String
```

The key code value for the SOFT1 soft key. The SOFT1 key code always corresponds to the left soft key; the SOFT2 always corresponds to the right soft key.

Availability

Flash Lite 2.0

Example

The following example creates a listener that handles the left and right soft keys:

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
var keyCode = Key.getCode();
switch (keyCode) {
    case ExtendedKey.SOFT1:
    // Handle left soft key.
    break;
    case ExtendedKey.SOFT2:
    // Handle right soft key
    break;
    }
}
Key.addListener(myListener);
```

SOFT2 (ExtendedKey.SOFT2 property)

```
public static SOFT2 : String
```

The key code value for the SOFT2 soft key. The SOFT2 key code always corresponds to the right soft key; the SOFT1 key code always corresponds to the left soft key.

Availability

Flash Lite 2.0

See also

SOFT1 (ExtendedKey.SOFT1 property)

SOFT3 (ExtendedKey.SOFT3 property)

```
public static SOFT3 : String
```

The key code value for the SOFT3 soft key.

Availability

Flash Lite 2.0

SOFT4 (ExtendedKey.SOFT4 property)

```
public static SOFT4 : String
```

The key code value for the SOFT4 soft key.

Availability

Flash Lite 2.0

SOFT5 (ExtendedKey.SOFT5 property)

```
public static SOFT5 : String
```

The key code value for the SOFT5 soft key.

Availability

Flash Lite 2.0

SOFT6 (ExtendedKey.SOFT6 property)

```
public static SOFT6 : String
```

The key code value for the SOFT6 soft key.

Availability
Flash Lite 2.0

SOFT7 (ExtendedKey.SOFT7 property)

```
public static SOFT7 : String
```

The key code value for the SOFT7 soft key.

Availability
Flash Lite 2.0

SOFT8 (ExtendedKey.SOFT8 property)

```
public static SOFT8 : String
```

The key code value for the SOFT8 soft key.

Availability
Flash Lite 2.0

SOFT9 (ExtendedKey.SOFT9 property)

```
public static SOFT9 : String
```

The key code value for the SOFT9 soft key.

SOFT10 (ExtendedKey.SOFT10 property)

```
public static SOFT10 : String
```

The key code value for the SOFT10 soft key.

Availability
Flash Lite 2.0

SOFT11 (ExtendedKey.SOFT11 property)

```
public static SOFT11 : String
```

The key code value for the SOFT11 soft key.

Availability
Flash Lite 2.0

SOFT12 (ExtendedKey.SOFT12 property)

```
public static SOFT12 : String
```

The key code value for the SOFT12 soft key.

Availability
Flash Lite 2.0

Function

```
Object
    |
    +-Function
```

```
public dynamic class Function
extends Object
```

Both user-defined and built-in functions in ActionScript are represented by Function objects, which are instances of the Function class.

Availability
Flash Lite 2.0

Property summary
Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property) prototype (Object.prototype property), __resolve (Object.__resolve property)

Method summary

Modifiers	Signature	Description
	apply(thisObject:Object, [argArray:Array])	Specifies the value of thisObject to be used within any function that ActionScript calls.
	call(thisObject:Object, [parameter1:Object])	Invokes the function represented by a Function object.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

apply (Function.apply method)

```
public apply(thisObject:Object, [argArray:Array])
```

Specifies the value of `thisObject` to be used within any function that ActionScript calls. This method also specifies the parameters to be passed to any called function. Because `apply()` is a method of the Function class, it is also a method of every Function object in ActionScript.

The parameters are specified as an Array object, unlike `Function.call()`, which specifies parameters as a comma-delimited list. This is often useful when the number of parameters to be passed is not known until the script actually executes.

Returns the value that the called function specifies as the return value.

Availability
Flash Lite 2.0

Parameters
thisObject : Object - The object to which myFunction is applied.

argArray : Array [optional] - An array whose elements are passed to myFunction as parameters.

Returns
Any value that the called function specifies.

Example
The following function invocations are equivalent:

```
Math.atan2(1, 0)
Math.atan2.apply(null, [1, 0])
```

The following example shows how `apply()` passes an array of parameters:

```
function theFunction() {
    trace(arguments);
}

// create a new array to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
theFunction.apply(null,firstArray);
// outputs: 1,2,3

// create a second array to pass as a parameter to apply()
var secondArray:Array = new Array("a", "b", "c");
theFunction.apply(null,secondArray);
// outputs a,b,c
```

The following example shows how `apply()` passes an array of parameters and specifies the value of this:

```
// define a function
function theFunction() {
    trace("this == myObj? " + (this == myObj));
    trace("arguments: " + arguments);
}

// instantiate an object
var myObj:Object = new Object();

// create arrays to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
var secondArray:Array = new Array("a", "b", "c");

// use apply() to set the value of this to be myObj and send firstArray
theFunction.apply(myObj,firstArray);
// output:
// this == myObj? true
// arguments: 1,2,3

// use apply() to set the value of this to be myObj and send secondArray
theFunction.apply(myObj,secondArray);
// output:
// this == myObj? true
// arguments: a,b,c
```

See also

call (Function.call method)

call (Function.call method)

```
public call(thisObject:Object, [parameter1:Object])
```

Invokes the function represented by a Function object. Every function in ActionScript is represented by a Function object, so all functions support this method.

In almost all cases, the function call (`()`) operator can be used instead of this method. The function call operator produces code that is concise and readable. This method is primarily useful when the `thisObject` parameter of the function invocation needs to be explicitly controlled. Normally, if a function is invoked as a method of an object, within the body of the function, `thisObject` is set to `myObject`, as shown in the following example:

```
myObject.myMethod(1, 2, 3);
```

In some situations, you might want `thisObject` to point somewhere else; for example, if a function must be invoked as a method of an object, but is not actually stored as a method of that object:

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

You can pass the value null for the `thisObject` parameter to invoke a function as a regular function and not as a method of an object. For example, the following function invocations are equivalent:

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

Returns the value that the called function specifies as the return value.

Availability

Flash Lite 2.0

Parameters

thisObject : Object - An object that specifies the value of `thisObject` within the function body.

parameter1 : Object [optional] - A parameter to be passed to the myFunction. You can specify zero or more parameters.

Example

The following example uses `Function.call()` to make a function behave as a method of another object, without storing the function in the object:

```
function myObject() {
}
function myMethod(obj) {
    trace("this == obj? " + (this == obj));
}
var obj:Object = new myObject();
myMethod.call(obj, obj);
```

The `trace()` statement displays:

```
this == obj? true
```

See also

`apply (Function.apply method)`

Key

```
Object
    |
    +-Key
```

```
public class Key
extends Object
```

The Key class is a top-level class whose methods and properties you can use without a constructor. Use the methods of the Key class to build interfaces. The properties of the Key class are constants representing the keys most commonly used to control applications, such as Arrow keys, Page Up, and Page Down. Use the System.capabilities properties to determine which keys a device supports.

Not all devices and Flash Lite content types support all keys. For example, devices that support two-way navigation don't support the left and right navigation keys. Also, not all devices have access to a device's soft keys. For information, see *Developing Flash Lite 2.x and 3.x Applications*.

Availability

Flash Lite 2.0

See also

`ExtendedKey`

"`has4WayKeyAS (capabilities.has4WayKeyAS property)`" on page 244

"`hasMappableSoftKeys (capabilities.hasMappableSoftKeys property)`" on page 248

"`hasQWERTYKeyboard (capabilities.hasQWERTYKeyboard property)`" on page 250

"softKeyCount (capabilities.softKeyCount property)" on page 258

Property summary

Modifiers	Property	Description
static	BACKSPACE : Number	The key code value for the Backspace key (8).
static	CAPSLOCK : Number	The key code value for the Caps Lock key (20).
static	CONTROL : Number	The key code value for the Control key (17).
static	DELETEKEY : Number	The key code value for the Delete key (46).
static	DOWN : Number	The key code value for the Down Arrow key (40).
static	END : Number	The key code value for the End key (35).
static	ENTER : Number	The key code value for the Enter key (13).
static	ESCAPE : Number	The key code value for the Escape key (27).
static	HOME : Number	The key code value for the Home key (36).
static	INSERT : Number	The key code value for the Insert key (45).
static	LEFT : Number	The key code value for the Left Arrow key (37).
static	_listeners : Array [read-only]	A list of references to all listener objects registered with the Key object.
static	PGDN : Number	The key code value for the Page Down key (34).
static	PGUP : Number	The key code value for the Page Up key (33).
static	RIGHT : Number	The key code value for the Right Arrow key (39).
static	SHIFT : Number	The key code value for the Shift key (16).
static	SPACE : Number	The key code value for the Spacebar (32).
static	TAB : Number	The key code value for the Tab key (9).
static	UP : Number	The key code value for the Up Arrow key (38).

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Event summary

Event	Description
onKeyDown = function() {}	Notified when a key is pressed.
onKeyUp = function() {}	Notified when a key is released.

Method summary

Modifiers	Signature	Description
static	addListener(listener: Object) : Void	Registers an object to receive onKeyDown and onKeyUp notification.
static	getAscii() : Number	Returns the ASCII code of the last key pressed or released.
static	getCode() : Number	Returns the key code value of the last key pressed.
static	isDown(code:Number) : Boolean	Returns true if the key specified in code is pressed; false otherwise.
static	removeListener(listener:Object) : Boolean	Removes an object previously registered with Key.addListener().

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

addListener (Key.addListener method)

```
public static addListener(listener:Object) : Void
```

Registers an object to receive onKeyDown and onKeyUp notifications. When a key is pressed or released, regardless of the input focus, all listening objects registered with addListener() have either their onKeyDown method or their onKeyUp method invoked. Multiple objects can listen for keyboard notifications.

Availability

Flash Lite 2.0

Parameters

listener: Object - An object with onKeyDown and onKeyUp methods.

Example

The following example creates a new listener object and defines functions for onKeyDown and onKeyUp. The last line calls addListener() to register the listener with the Key object so that it can receive notification from the key down and key up events.

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
    trace ("You pressed a key.");
}
myListener.onKeyUp = function () {
    trace ("You released a key.");
}
Key.addListener(myListener);
```

See also

getCode (Key.getCode method), isDown (Key.isDown method), onKeyDown (Key.onKeyDown event listener), onKeyUp (Key.onKeyUp event listener), removeListener (Key.removeListener method)

BACKSPACE (Key.BACKSPACE property)

```
public static BACKSPACE : Number
```

The key code value for the Backspace key (8).

Availability

Flash Lite 2.0

Example

The following example creates a new listener object and defines a function for `onKeyDown`. The last line uses `addListener()` to register the listener with the Key object so that it can receive notification from the key down event.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.BACKSPACE)) {
    trace("you pressed the Backspace key.");
    } else {
    trace("you DIDN'T press the Backspace key.");
    }
};
Key.addListener(keyListener);
```

CAPSLOCK (Key.CAPSLOCK property)

```
public static CAPSLOCK : Number
```

The key code value for the Caps Lock key (20).

Availability

Flash Lite 2.0

CONTROL (Key.CONTROL property)

```
public static CONTROL : Number
```

The key code value for the Control key (17).

Availability

Flash Lite 2.0

DELETEKEY (Key.DELETEKEY property)

```
public static DELETEKEY : Number
```

The key code value for the Delete key (46).

Availability

Flash Lite 2.0

DOWN (Key.DOWN property)

```
public static DOWN : Number
```

The key code value for the Down Arrow key (40).

Availability
Flash Lite 2.0

Example
The following example moves a movie clip called `car_mc` a constant distance (10) when you press the arrow keys. Place any movie clip on the Stage and give it the instance name car_mc.

```
var DISTANCE:Number = 10;
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
    case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
    case Key.UP :
    car_mc._y -= DISTANCE;
    break;
    case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
    case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
    }
};
Key.addListener(keyListener_obj);
```

END (Key.END property)

```
public static END : Number
```

The key code value for the End key (35).

Availability
Flash Lite 2.0

ENTER (Key.ENTER property)

```
public static ENTER : Number
```

The key code value for the Enter key (13).

Availability
Flash Lite 2.0

Example
The following example moves a movie clip when you press the arrow keys. The movie clip stops when you press Select and delete the `onEnterFrame` event. Place any movie clip on the Stage and give it the instance name car_mc.

```
var DISTANCE:Number = 5;
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    switch (Key.getCode()) {
    case Key.LEFT :
    car_mc.onEnterFrame = function() {
        this._x -= DISTANCE;
    };
    break;
    case Key.UP :
    car_mc.onEnterFrame = function() {
        this._y -= DISTANCE;
    };
    break;
    case Key.RIGHT :
    car_mc.onEnterFrame = function() {
        this._x += DISTANCE;
    };
    break;
    case Key.DOWN :
    car_mc.onEnterFrame = function() {
        this._y += DISTANCE;
    };
    break;
    case Key.ENTER :
    delete car_mc.onEnterFrame;
    break;
    }
};
Key.addListener(keyListener);
```

ESCAPE (Key.ESCAPE property)

```
public static ESCAPE : Number
```

The key code value for the Escape key (27).

Availability

Flash Lite 2.0

Example

The following example sets a timer. When you press Select, the Output panel displays how long it took you to press the key.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.ENTER)) {
    // get the current timer, convert the value
    // to seconds and round it to two decimal places.
        var timer:Number = Math.round(getTimer()/10)/100;
        trace("You pressed the Select key after: "+getTimer()+"ms ("+timer+"s)");
    }
};
Key.addListener(keyListener);
```

getAscii (Key.getAscii method)

```
public static getAscii() : Number
```

Returns the ASCII code of the last key pressed or released. The ASCII values returned are English keyboard values. For example, if you press Shift+2 on either a Japanese or English keyboard, `Key.getAscii()` returns `@`.

Availability

Flash Lite 2.0

Returns

Number - The ASCII value of the last key pressed. This method returns 0 if no key was pressed or released, or if the ASCII value is not accessible for security reasons.

Example

The following example calls the `getAscii()` method any time a key is pressed. The example creates a listener object named `keyListener` and defines a function that responds to the `onKeyDown` event by calling `Key.getAscii()`. The keyListener object is then registered to the Key object, which broadcasts the `onKeyDown` message whenever a key is pressed while the SWF file plays.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("The ASCII code for the last key typed is: "+Key.getAscii());
};
Key.addListener(keyListener);
```

The following example adds a call to `Key.getAscii()` to show how `getAscii()` and `getCode()` differ. The main difference is that `Key.getAscii()` differentiates between uppercase and lowercase letters, and `Key.getCode()` does not.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("For the last key typed:");
    trace("\tThe Key code is: "+Key.getCode());
    trace("\tThe ASCII value is: "+Key.getAscii());
    trace("");
};
Key.addListener(keyListener);
```

getCode (Key.getCode method)

```
public static getCode() : Number
```

Returns the key code value of the last key pressed.

The Flash Lite implementation of this method returns a string or a number, depending on the key code passed in by the platform. The only valid key codes are the standard key codes accepted by this class and the special key codes listed as properties of the ExtendedKey class.

Availability

Flash Lite 2.0

Returns

Number - The key code of the last key pressed. This method returns 0 if no key was pressed or released, or if the key code is not accessible for security reasons.

Example

The following example calls the `getCode()` method any time a key is pressed. The example creates a listener object named `keyListener` and defines a function that responds to the `onKeyDown` event by calling `Key.getCode()`. The keyListener object is registered to the Key object, which broadcasts the `onKeyDown` message whenever a key is pressed while the SWF file plays.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    // Compare return value of getCode() to constant
    if (Key.getCode() == Key.ENTER) {
        trace ("Virtual key code: "+Key.getCode()+" (ENTER key)");
    }
    else {
        trace("Virtual key code: "+Key.getCode());
    }
};
Key.addListener(keyListener);
```

The following example adds a call to `Key.getAscii()` to show how the two methods differ. The main difference is that `Key.getAscii()` differentiates between uppercase and lowercase letters, and `Key.getCode()` does not.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("For the last key typed:");
    trace("\tThe Key code is: "+Key.getCode());
    trace("\tThe ASCII value is: "+Key.getAscii());
    trace("");
};
Key.addListener(keyListener);
```

Availability

Flash Lite 2.0

See also

getAscii (Key.getAscii method)

HOME (Key.HOME property)

```
public static HOME : Number
```

The key code value for the Home key (36).

Availability

Flash Lite 2.0

INSERT (Key.INSERT property)

```
public static INSERT : Number
```

The key code value for the Insert key (45).

Availability

Flash Lite 2.0

isDown (Key.isDown method)

```
public static isDown(code:Number) : Boolean
```

Returns `true` if the key specified in `code` is pressed; `false` otherwise.

Availability

Flash Lite 2.0

Parameters

code : Number - The key code value assigned to a specific key or a Key class property associated with a specific key.

Returns

Boolean - The value `true` if the key specified in `code` is pressed; `false` otherwise.

Example

The following script lets the user use the Left and Right keys to control the location of a movie clip on the Stage called `car_mc`:

```
car_mc.onEnterFrame = function() {
    if (Key.isDown(Key.RIGHT)) {
    this._x += 10;
    } else if (Key.isDown(Key.LEFT)) {
    this._x -= 10;
    }
};
```

LEFT (Key.LEFT property)

```
public static LEFT : Number
```

The key code value for the Left Arrow key (37).

Availability

Flash Lite 2.0

_listeners (Key._listeners property)

```
public static _listeners : Array [read-only]
```

A list of references to all listener objects registered with the Key object. This property is intended for internal use, but may be useful if you want to ascertain the number of listeners currently registered with the Key object. Objects are added and removed from this array by calls to the addListener() and removeListener() methods.

Availability

Flash Lite 2.0

Example

The following example shows how to use the length property to ascertain the number of listener objects currently registered to the Key object.

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
    trace ("You pressed a key.");
}
Key.addListener(myListener);

trace(Key._listeners.length); // Output: 1
```

onKeyDown (Key.onKeyDown event listener)

```
onKeyDown = function() {}
```

Notified when a key is pressed. To use `onKeyDown`, you must create a listener object. You can then define a function for `onKeyDown` and use `addListener()` to register the listener with the Key object, as shown in the following example:

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("DOWN -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
"+chr(Key.getAscii()));
};
keyListener.onKeyUp = function() {
    trace("UP -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
"+chr(Key.getAscii()));
};
Key.addListener(keyListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

Availability

Flash Lite 2.0

See also

`addListener (Key.addListener method)`

onKeyUp (Key.onKeyUp event listener)

```
onKeyUp = function() {}
```

Notified when a key is released. To use `onKeyUp`, you must create a listener object. You can then define a function for `onKeyUp` and use `addListener()` to register the listener with the Key object, as shown in the following example:

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("DOWN -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
"+chr(Key.getAscii()));
};
keyListener.onKeyUp = function() {
    trace("UP -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
"+chr(Key.getAscii()));
};
Key.addListener(keyListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

Availability
Flash Lite 2.0

See also
`addListener (Key.addListener method)`

PGDN (Key.PGDN property)

```
public static PGDN : Number
```

The key code value for the Page Down key (34).

Availability
Flash Lite 2.0

PGUP (Key.PGUP property)

```
public static PGUP : Number
```

The key code value for the Page Up key (33).

Availability
Flash Lite 2.0

removeListener (Key.removeListener method)

```
public static removeListener(listener:Object) : Boolean
```

Removes an object previously registered with `Key.addListener()`.

Availability
Flash Lite 2.0

Parameters
listener : Object - An object.

Returns
Boolean - If *listener* was successfully removed, the method returns `true`. If *listener* was not successfully removed (for example, because *listener* was not on the Key objects listener list), the method returns `false`.

RIGHT (Key.RIGHT property)

```
public static RIGHT : Number
```

The key code value for the Right Arrow key (39).

Availability
Flash Lite 2.0

Example
The following example moves a movie clip on the Stage called `car_mc` when you press the arrow keys.

```
var DISTANCE:Number = 10;
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
    case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
    case Key.UP :
    car_mc._y -= DISTANCE;
    break;
    case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
    case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
    }
};
Key.addListener(keyListener_obj);
```

SHIFT (Key.SHIFT property)

```
public static SHIFT : Number
```

The key code value for the Shift key (16).

Availability
Flash Lite 2.0

SPACE (Key.SPACE property)

```
public static SPACE : Number
```

The key code value for the Spacebar (32).

Availability
Flash Lite 2.0

TAB (Key.TAB property)

```
public static TAB : Number
```

The key code value for the Tab key (9).

Availability
Flash Lite 2.0

UP (Key.UP property)

```
public static UP : Number
```

The key code value for the Up Arrow key (38).

Availability
Flash Lite 2.0

Example
The following example moves a movie clip on the Stage called `car_mc` a constant distance (10) when you press the arrow keys.

```
var DISTANCE:Number = 10;
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
    case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
    case Key.UP :
    car_mc._y -= DISTANCE;
    break;
    case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
    case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
    }
};
Key.addListener(keyListener_obj);
```

LoadVars

```
Object
    |
    +-LoadVars

public dynamic class LoadVars
extends Object
```

The LoadVars class is an alternative to the loadVariables() function for transferring variables between a Flash Lite and a web server over HTTP. Use the LoadVars class to obtain verification of successful data loading and to monitor download progress.

The LoadVars class lets you send all the variables in an object to a specified URL and load all the variables at a specified URL into an object. It also lets you send specific variables, rather than all the variables, which can make your application more efficient. Use the LoadVars.onLoad handler to ensure that your application runs when data is loaded, and not before.

The LoadVars class works much like the XML class; it uses the methods `load()`, `send()`, and `sendAndLoad()` to communicate with a server. The main difference between the LoadVars class and the XML class is that LoadVars transfers ActionScript name and value pairs, rather than an XML DOM tree stored in the XML object. The LoadVars class follows the same security restrictions as the XML class.

Availability
Flash Lite 2.0

See also

`loadVariables function`, `onLoad (LoadVars.onLoad handler)`, `hasXMLSocket (capabilities.hasXMLSocket property)`

Property summary

Modifiers	Property	Description
	contentType : String	The MIME type that is sent to the server when you call `LoadVars.send()` or `LoadVars.sendAndLoad()`.
	loaded : Boolean	A Boolean value that indicates whether a `load` or `sendAndLoad` operation has completed, `undefined` by default.

Properties inherited from class Object

`constructor (Object.constructor property)`, `__proto__ (Object.__proto__ property)` `prototype (Object.prototype property)`, `__resolve (Object.__resolve property)`

Event summary

Event	Description
onData = `function(src:String) {}`	Invoked when data has completely downloaded from the server or when an error occurs while data is downloading from a server.
onLoad = `function(success:Boolean) {}`	Invoked when a `LoadVars.load()` or `LoadVars.sendAndLoad()` operation has ended.

Constructor summary

Signature	Description
LoadVars `()`	Creates a LoadVars object.

Method summary

Modifiers	Signature	Description
	addRequestHeader `(header:Object, headerValue:String) : Void`	Adds or changes HTTP request headers (such as `Content-Type` or `SOAPAction`) sent with `POST` actions.
	decode `(queryString:String) : Void`	Converts the variable string to properties of the specified LoadVars object.
	getBytesLoaded `() :` Number	Returns the number of bytes downloaded by `LoadVars.load()` or `LoadVars.sendAndLoad()`.
	getBytesTotal `() :` Number	Returns the total number of bytes downloaded by `LoadVars.load()` or `LoadVars.sendAndLoad()`.
	load `(url:String) :` Boolean	Downloads variables from the specified URL, parses the variable data, and places the resulting variables into *the LoadVars object*.

Modifiers	Signature	Description
	send(url:String, target:String, [method:String]) : Boolean	Sends the variables in the *LoadVars* object to the specified URL.
	sendAndLoad(url:String, target:Object, [method:String]) : Boolean	Posts variables in the *LoadVars* object to the specified URL.
	toString() : String	Returns a string containing all enumerable variables in *the LoadVars object*, in the MIME content encoding *application/x-www-form-urlencoded*.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

addRequestHeader (LoadVars.addRequestHeader method)

```
public addRequestHeader(header:Object, headerValue:String) : Void
```

Adds or changes HTTP request headers (such as `Content-Type` or `SOAPAction`) sent with `POST` actions. In the first usage, you pass two strings to the method: `header` and `headerValue`. In the second usage, you pass an array of strings, alternating header names and header values.

If multiple calls are made to set the same header name, each successive value will replace the value set in the previous call.

The following standard HTTP headers *cannot* be added or changed with this method: `Accept-Ranges`, `Age`, `Allow`, `Allowed`, `Connection`, `Content-Length`, `Content-Location`, `Content-Range`, `ETag`, `Host`, `Last-Modified`, `Locations`, `Max-Forwards`, `Proxy-Authenticate`, `Proxy-Authorization`, `Public`, `Range`, `Retry-After`, `Server`, `TE`, `Trailer`, `Transfer-Encoding`, `Upgrade`, `URI`, `Vary`, `Via`, `Warning`, and `WWW-Authenticate`.

Availability

Flash Lite 2.0

Parameters

header : Object - A string or array of strings that represents an HTTP request header name.

headerValue : String - A string that represents the value associated with `header`.

Example

The following example adds a custom HTTP header named `SOAPAction` with a value of `Foo` to the my_lv object:

```
my_lv.addRequestHeader("SOAPAction", "'Foo'");
```

The following example creates an array named `headers` that contains two alternating HTTP headers and their associated values. The array is passed as an argument to `addRequestHeader()`.

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];
my_lv.addRequestHeader(headers);
```

The following example creates a new LoadVars object that adds a request header called `FLASH-UUID`. The header contains a variable that can be checked by the server.

```
var my_lv:LoadVars = new LoadVars();
my_lv.addRequestHeader("FLASH-UUID", "41472");
my_lv.name = "Mort";
my_lv.age = 26;
my_lv.send("http://flash-mx.com/mm/cgivars.cfm", "_blank", "POST");
```

See also

`addRequestHeader (XML.addRequestHeader method)`

contentType (LoadVars.contentType property)

```
public contentType : String
```

The MIME type that is sent to the server when you call `LoadVars.send()` or `LoadVars.sendAndLoad()`. The default is *application/x-www-form-urlencoded*.

Availability

Flash Lite 2.0

Example

The following example creates a LoadVars object and displays the default content type of the data that is sent to the server.

```
var my_lv:LoadVars = new LoadVars();
trace(my_lv.contentType); // output: application/x-www-form-urlencoded
```

See also

`send (LoadVars.send method)`, `sendAndLoad (LoadVars.sendAndLoad method)`

decode (LoadVars.decode method)

```
public decode(queryString:String) : Void
```

Converts the variable string to properties of the specified LoadVars object.

This method is used internally by the `LoadVars.onData` event handler. Most users do not need to call this method directly. If you override the `LoadVars.onData` event handler, you can explicitly call `LoadVars.decode()` to parse a string of variables.

Availability

Flash Lite 2.0

Parameters

queryString: String - A URL-encoded query string containing name/value pairs.

Example

The following example traces the three variables:

```
// Create a new LoadVars object
var my_lv:LoadVars = new LoadVars();
//Convert the variable string to properties
my_lv.decode("name=Mort&score=250000");
trace(my_lv.toString());
// Iterate over properties in my_lv
for (var prop in my_lv) {
    trace(prop+" -> "+my_lv[prop]);
}
```

See also

onData (LoadVars.onData handler), parseXML (XML.parseXML method)

getBytesLoaded (LoadVars.getBytesLoaded method)

```
public getBytesLoaded() : Number
```

Returns the number of bytes downloaded by a call to `LoadVars.load()` or `LoadVars.sendAndLoad()`. This method returns `undefined` if no load operation is in progress or if a load operation has not yet begun.

Note: *You cannot use this method to return information about a local file on your hard disk.*

Availability

Flash Lite 2.0

Returns

Number - An integer.

See also

load (LoadVars.load method), sendAndLoad (LoadVars.sendAndLoad method)

getBytesTotal (LoadVars.getBytesTotal method)

```
public getBytesTotal() : Number
```

Returns the total number of bytes downloaded by `LoadVars.load()` or `LoadVars.sendAndLoad()`. This method returns `undefined` if no load operation is in progress or if a load operation has not started. This method also returns `undefined` if the number of total bytes can't be determined (for example, if the download was initiated but the server did not transmit an HTTP content-length).

Note: *You cannot use this method to return information about a local file on your hard disk.*

Availability

Flash Lite 2.0

Returns

Number - An integer.

See also

load (LoadVars.load method), sendAndLoad (LoadVars.sendAndLoad method)

load (LoadVars.load method)

```
public load(url:String) : Boolean
```

Downloads variables from the specified URL, parses the variable data, and places the resulting variables into *a LoadVars object*. Any properties in the LoadVars object with the same names as downloaded variables are overwritten. Any properties in the LoadVars object with different names than downloaded variables are not deleted. This is an asynchronous action.

The downloaded data must be in the MIME content type *application/x-www-form-urlencoded*.

This is the same format used by `loadVariables()`.

In SWF files running in a version of the player earlier than Flash Player 7, `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the left-most component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com because both files are in the same superdomain named someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `url` must be in exactly the same domain. For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file.

Also, in files published for Flash Player 7, case-sensitivity is supported for external variables loaded with `LoadVars.load()`.

This method is similar to `XML.load()`.

Availability

Flash Lite 2.0

Parameters

url : String - The URL from which to download the variables. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see the Description section.

Returns

Boolean - If no parameter (null) is passed, `false` ; otherwise, `true` . Use the `onLoad()` event handler to check the success of loaded data.

Example

The following code defines an `onLoad` handler that signals when data is returned to the application from a text file, then loads the data from the text file and sends it to the Output window.

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
    if (success) {
        trace(this.toString());
    } else {
        trace("Error loading/parsing LoadVars.");
    }
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

See also

load (XML.load method), loaded (LoadVars.loaded property), onLoad (LoadVars.onLoad handler)

loaded (LoadVars.loaded property)

```
public loaded : Boolean
```

A Boolean value that indicates whether a `load` or `sendAndLoad` operation has completed, `undefined` by default. When a `LoadVars.load()` or `LoadVars.sendAndLoad()` operation is started, the `loaded` property is set to `false`; when the operation completes, the `loaded` property is set to `true`. If the operation has not completed or has failed with an error, the `loaded` property remains set to `false`.

This property is similar to the `XML.loaded`property.

Availability

Flash Lite 2.0

Example

The following example loads a text file and displays information in the Output panel when the operation completes.

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
    trace("LoadVars loaded successfully: "+this.loaded);
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

See also

`load (LoadVars.load method)`, `sendAndLoad (LoadVars.sendAndLoad method)`, `load (XML.load method)`

LoadVars constructor

```
public LoadVars()
```

Creates a LoadVars object. Call the methods of that LoadVars object to send and load data.

Availability

Flash Lite 2.0

Example

The following example creates a LoadVars object called `my_lv`:

```
var my_lv:LoadVars = new LoadVars();
```

onData (LoadVars.onData handler)

```
onData = function(src:String) {}
```

Invoked when data has completely downloaded from the server or when an error occurs while data is downloading from a server. This handler is invoked before the data is parsed and can be used to call a custom parsing routine instead of the one built in to Flash Lite. The value of the `src` parameter passed to the function assigned to `LoadVars.onData` can be either `undefined` or a string that contains the URL-encoded name-value pairs downloaded from the server. If the `src` parameter is `undefined`, an error occurred while downloading the data from the server.

The default implementation of `LoadVars.onData` invokes `LoadVars.onLoad`. You can override this default implementation by assigning a custom function to `LoadVars.onData`, but `LoadVars.onLoad` is not called unless you call it in your implementation of `LoadVars.onData`.

Availability
Flash Lite 2.0

Parameters
src : String - A string or `undefined`; the raw (unparsed) data from a `LoadVars.load()` or `LoadVars.sendAndLoad()` method call.

Example
The following example loads a text file and displays content in a TextField instance called `content_txt` when the operation completes. If an error occurs, information displays in the Output panel.

```
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src == undefined) {
        trace("Error loading content.");
        return;
    }
    content_txt.text = src;
};
my_lv.load("http://www.helpexamples.com/flash/params.txt", my_lv, "GET");
```

See also
`onLoad (LoadVars.onLoad handler)`, `load (LoadVars.load method)`, `sendAndLoad (LoadVars.sendAndLoad method)`

onLoad (LoadVars.onLoad handler)

```
onLoad = function(success:Boolean) {}
```

Invoked when a `LoadVars.load()` or `LoadVars.sendAndLoad()` operation has ended. If the operation was successful, *my_lv* is populated with variables downloaded by the operation, and these variables are available when this handler is invoked.

This handler is `undefined` by default.

This event handler is similar to `XML.onLoad`.

Availability
Flash Lite 2.0

Parameters
success : Boolean - Indicates whether the load operation ended in success (`true`) or failure (`false`).

Example
See the example for the LoadVars.sendAndLoad() method.

See also
`onLoad (XML.onLoad handler)`, `loaded (LoadVars.loaded property)`, `load (LoadVars.load method)`, `sendAndLoad (LoadVars.sendAndLoad method)`

send (LoadVars.send method)

```
public send(url:String, target:String, [method:String]) : Boolean
```

Sends the variables in the *LoadVars* object to the specified URL. Variables are concatenated into a string in the *application/x-www-form-urlencoded* format, or the value of LoadVars.contentType. The `POST` method is used unless `GET` is specified.

You must specify the `target` parameter to execute that the script or application at the specified URL. If you omit the `target` parameter, the function returns `true`, but the script or application is not executed.

The `send()` method is useful if you want the server response to:

- Replace the SWF content (use `"_self"` as the `target` parameter);
- Appear in a new window (use `"_blank"` as the `target` parameter);
- Appear in the parent or top-level frame (use `"_parent"` or `"_top"` as the `target` parameter);
- Appear in a named frame (use the frame's name as a string for the `target` parameter).

A successful `send()` method call always opens a new browser window or replaces content in an existing window or frame. If you would rather send information to a server and continue playing your SWF file without opening a new window or replacing content in a window or frame, use the `LoadVars.sendAndLoad()` method.

This method is similar to `XML.send()`.

Availability

Flash Lite 2.0

Parameters

url : String - The URL to which to upload variables.

target : String - The browser window or frame in which a response appears. You can enter the name of a specific window or select from the following reserved target names:

- `"_self"` specifies the current frame in the current window.
- `"_blank"` specifies a new window.
- `"_parent"` specifies the parent of the current frame.
- `"_top"` specifies the top-level frame in the current window.

method : String (optional) - The `GET` or `POST` method of the HTTP protocol. The default value is `POST`.

Returns

Boolean - If no parameters are specified, false, otherwise, true.

Example

The following example copies two values from text fields and sends the data to a CFM script, which is used to handle the information. For example, the script might check if the user got a high score and then insert that data into a database table.

```
var my_lv:LoadVars = new LoadVars();
my_lv.playerName = playerName_txt.text;
my_lv.playerScore = playerScore_txt.text;
my_lv.send("setscore.cfm", "_blank", "POST");
```

See also

`sendAndLoad (LoadVars.sendAndLoad method)`, `send (XML.send method)`

sendAndLoad (LoadVars.sendAndLoad method)

```
public sendAndLoad(url:String, target:Object, [method:String]) : Boolean
```

Posts variables in the *LoadVars* object to the specified URL. The server response is downloaded, parsed, and the resulting variables are placed in the target object.

Variables are posted in the same manner as `LoadVars.send()`. Variables are downloaded into `target` in the same manner as `LoadVars.load()`.

In SWF files running in a version of the player earlier than Flash Player 7 (i.e. Flash Lite 1.x), `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the left-most component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later (i.e. Flash Lite 2.x and 3.x), `url` must be in exactly the same domain. For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file.

This method is similar to `XML.sendAndLoad()`.

Availability

Flash Lite 2.0

Parameters

url : String - The URL to which to upload variables. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file.

target : Object - The LoadVars or XML object that receives the downloaded variables.

method : String (optional) - The `GET` or `POST` method of the HTTP protocol. The default value is `POST`.

Returns

Boolean

Example

For the following example, add an Input text field called `name_txt`, a Dynamic text field called `result_txt`, and a button called `submit_btn` to the Stage. When the user clicks the button, two LoadVars objects are created: `send_lv` and `result_lv`. The send_lv object copies the name from the `name_txt` instance and sends the data to greeting.cfm. The result from this script loads into the result_lv object, and the server response displays in the result_txt text field. Add the following ActionScript to Frame 1 of the Timeline:

```
var send_lv:LoadVars = new LoadVars();
var result_lv:LoadVars = new LoadVars();
result_lv.onLoad = function(success:Boolean) {
    if (success) {
        result_txt.text = result_lv.welcomeMessage;
    } else {
        result_txt.text = "Error connecting to server.";
    }
};

submit_btn.onRelease = function(){
    send_lv.name = name_txt.text;
    send_lv.sendAndLoad("http://www.flash-mx.com/mm/greeting.cfm", result_lv);
}
```

See also

send (LoadVars.send method), load (LoadVars.load method), sendAndLoad (XML.sendAndLoad method)

toString (LoadVars.toString method)

```
public toString() : String
```

Returns a string containing all enumerable variables in the LoadVars object, in the MIME content encoding *application/x-www-form-urlencoded*.

Availability

Flash Lite 2.0

Returns

String

Example

The following example instantiates a new `LoadVars()` object, creates two properties, and uses `toString()` to return a string containing both properties in URL encoded format:

```
var my_lv:LoadVars = new LoadVars();
my_lv.name = "Gary";
my_lv.age = 26;
trace (my_lv.toString()); //output: age=26&name=Gary
```

LocalConnection

```
Object
    |
    +-LocalConnection

public dynamic class LocalConnection
extends Object
```

The LocalConnection class lets you develop SWF files that can send instructions to each other without the use of `fscommand()` or JavaScript. LocalConnection objects can communicate only among SWF files that are running on the same client device, but they can be running in different applications. You can use LocalConnection objects to send and receive data within a single SWF file, but this is not a standard implementation; all the examples in this section illustrate communication between different SWF files.

Use the `LocalConnection.send()` and `LocalConnection.connect()` methods to send and receive data. Notice that both the `LocalConnection.send()` and `LocalConnection.connect()` commands specify the same connection name, `lc_name`:

```
// Code in the receiving SWF file
this.createTextField("result_txt", 1, 10, 10, 100, 22);
result_txt.border = true;
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.methodToExecute = function(param1:Number, param2:Number) {
    result_txt.text = param1+param2;
};
receiving_lc.connect("lc_name");

// Code in the sending SWF file
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", 5, 7);
```

The simplest way to use a LocalConnection object is to allow communication only between LocalConnection objects located in the same domain because you won't have security issues. However, if you need to allow communication between domains, you have several ways to implement security measures. For more information, see the discussion of the `connectionName` parameter in `LocalConnection.send()` and the `LocalConnection.allowDomain` and `LocalConnection.domain()` entries.

Availability

Flash Lite 3.1

Property summary

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Event summary

Event	Description
allowDomain = function([sendingDomain:]) {} String	Invoked whenever a LocalConnection object receives a request to invoke a method from another LocalConnection object.
allowInsecureDomain = function([sendingDomain:]) {} String	Invoked whenever a receiving LocalConnection object, which is in a SWF file hosted at a domain using a secure protocol (HTTPS), receives a request to invoke a method from a sending LocalConnection object that is in a SWF file hosted at a nonsecure protocol.
onStatus = function(infoObject:Object) {}	Invoked after a sending LocalConnection object tries to send a command to a receiving LocalConnection object.

Constructor summary

Signature	Description
LocalConnection()	Creates a LocalConnection object.

Method summary

Modifiers	Signature	Description
	close() : Void	Closes (disconnects) a LocalConnection object.
	connect(connectionName:String) :Boolean	Prepares a LocalConnection object to receive commands from a LocalConnection.send() command (called the *sending LocalConnection object*).
	domain() : String	Returns a string representing the domain of the location of the current SWF file.
	send(connectionName:String, methodName:String, [args: Object]) :Boolean	Invokes the method named method on a connection opened with the LocalConnection.connect(connectionName) command (the receiving LocalConnection object).

addProperty (Object.addProperty method),hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method),isPrototypeOf (Object.isPrototypeOf method),registerClass (Object.registerClass method),toString (Object.toString method),unwatch (Object.unwatch method),valueOf (Object.valueOf method),watch (Object.watch method)

allowDomain (LocalConnection.allowDomain handler)

```
allowDomain = function([sendingDomain:String]) {}
```

Invoked whenever receiving_lc receives a request to invoke a method from a sending LocalConnection object. Flash expects the code you implement in this handler to return a Boolean value of true or false. If the handler doesn't return true, the request from the sending object is ignored, and the method is not invoked.

When this event handler is absent, Flash Lite player applies a default security policy, which is equivalent to the following code:

```
my_lc.allowDomain = function (sendingDomain)
{
    return (sendingDomain == this.domain());
}
```

Use LocalConnection.allowDomain to explicitly permit LocalConnection objects from specified domains, or from any domain, to execute methods of the receiving LocalConnection object. If you don't declare the sendingDomain parameter, you probably want to accept commands from any domain, and the code in your handler would be simply return true. If you do declare sendingDomain, you probably want to compare the value of sendingDomain with domains from which you want to accept commands. The following examples show both implementations.

In files authored for Flash Player 6 or earlier (i.e. Flash Lite 1.x), the sendingDomain parameter contains the superdomain of the caller. In files authored for Flash Player 7 or later (i.e. Flash Lite 2.x and 3.x), the sendingDomain parameter contains the exact domain of the caller. In the latter case, to allow access by SWF files hosted at either www.domain.com or store.domain.com, you must explicitly allow access from both domains.

```
// For Flash Player 6
receiving_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="domain.com");
}
// For Flash Player 7 or later
receiving_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="www.domain.com" ||
        sendingDomain=="store.domain.com");
}
```

Also, for files authored for Flash Player 7 or later (i.e. Flash Lite 2.x and 3.x), you can't use this method to let SWF files hosted using a secure protocol (HTTPS) allow access from SWF files hosted in nonsecure protocols; you must use the `LocalConnection.allowInsecureDomain` event handler instead.

Occasionally, you might encounter the following situation. Suppose you load a child SWF file from a different domain. You want to implement this method so that the child SWF file can make LocalConnection calls to the parent SWF file, but you don't know the final domain from which the child SWF file will come. This can happen, for example, when you use load-balancing redirects or third-party servers.

In this situation, you can use the `MovieClip._url` property in your implementation of this method. For example, if you load a SWF file into my_mc, you can then implement this method by checking whether the domain argument matches the domain of `my_mc._url`. (You must parse the domain out of the full URL contained in `my_mc._url`.)

If you do this, make sure that you wait until the SWF file in `my_mc` is loaded, because the _url property will not have its final, correct value until the file is completely loaded. The best way to determine when a child SWF file finishes loading is to use `MovieClipLoader.onLoadComplete`.

The opposite situation can also occur: You might create a child SWF file that wants to accept LocalConnection calls from its parent but doesn't know the domain of its parent. In this situation, implement this method by checking whether the domain argument matches the domain of _parent._url. Again, you must parse the domain out of the full URL from _parent._url. In this situation, you don't have to wait for the parent SWF file to load; the parent will already be loaded by the time the child loads.

Availability

Flash Lite 3.1

Parameters

sendingDomain : String [optional] - A string that specifies the domain of the SWF file that contains the sending LocalConnection object.

Example

The following example shows how a LocalConnection object in a receiving SWF file can permit SWF files from any domain to invoke its methods. Compare this to the example in `LocalConnection.connect()`, in which only SWF files from the same domain can invoke the `trace()` method in the receiving SWF file. For a discussion of the use of the underscore (_) in the connection name, see `LocalConnection.send()`.

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100, 20);
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain:String) {
    domain_txt.text = sendingDomain;
    return true;
};
my_lc.allowInsecureDomain = function(sendingDomain:String) {
    return (sendingDomain == this.domain());
}
my_lc.sayHello = function(name:String) {
    welcome_txt.text = "Hello, "+name;
};
my_lc.connect("_mylc");
```

The following example sends a string to the previous SWF file and displays a status message about whether the local connection was able to connect to the file. A TextInput component called `name_ti`, a TextArea instance called `status_ta` and a Button instance called `send_button` are used to display content.

```
var sending_lc:LocalConnection;
var sendListener:Object = new Object();
sendListener.click = function(evt:Object) {
    sending_lc = new LocalConnection();
    sending_lc.onStatus = function(infoObject:Object) {
    switch (infoObject.level) {
    case 'status' :
        status_ta.text = "LocalConnection connected successfully.";
        break;
    case 'error' :
        status_ta.text = "LocalConnection encountered an error.";
        break;
    }
    };
    sending_lc.send("_mylc", "sayHello", name_ti.text);
};
send_button.addEventListener("click", sendListener);
```

In the following example, the receiving SWF file, which resides in `thisDomain.com`, accepts commands only from SWF files located in `thisDomain.com` or `thatDomain.com`:

```
var aLocalConn:LocalConnection = new LocalConnection();
aLocalConn.Trace = function(aString) {
    aTextField += aString+newline;
};
aLocalConn.allowDomain = function(sendingDomain) {
    return (sendingDomain == this.domain() || sendingDomain == "www.macromedia.com");
};
aLocalConn.connect("_mylc");
```

When published for Flash Player 7 or later (Flash Lite 2.x and 3.x), exact domain matching is used. This means that the example will fail if the SWF files are located at www.thatDomain.com but will work if the files are located at thatDomain.com.

See also

connect (LocalConnection.connect method), domain (LocalConnection.domain method), send (LocalConnection.send method), _url (MovieClip._url property), onLoadComplete (MovieClipLoader.onLoadComplete event listener), _parent property

allowInsecureDomain (LocalConnection.allowInsecureDomain handler)

```
allowInsecureDomain = function([sendingDomain:String]) {}
```

Invoked whenever receiving_lc, which is in a SWF file hosted at a domain using a secure protocol (HTTPS), receives a request to invoke a method from a sending LocalConnection object that is in a SWF file hosted at a nonsecure protocol. Flash expects the code you implement in this handler to return a Boolean value of `true` or `false`. If the handler doesn't return true, the request from the sending object is ignored, and the method is not invoked.

By default, SWF files hosted using the HTTPS protocol can be accessed only by other SWF files hosted using the HTTPS protocol. This implementation maintains the integrity provided by the HTTPS protocol.

Using this method to override the default behavior is not recommended, as it compromises HTTPS security. However, you might need to do so, for example, if you need to permit access to HTTPS files published for Flash Player 7 or later (i.e. Flash Lite 2.x and 3.x) from HTTP files published for Flash Player 6.

A SWF file published for Flash Player 6 can use the `LocalConnection.allowDomain` event handler to permit HTTP to HTTPS access. However, because security is implemented differently in Flash Player 7, you must use the `LocalConnection.allowInsecureDomain()` method to permit such access in SWF files published for Flash Player 7 or later.

Availability

Flash Lite 3.1

Parameters

sendingDomain : String [optional] - A string that specifies the domain of the SWF file that contains the sending LocalConnection object.

Example

The following example allows connections from the current domain or from www.macromedia.com, or allows insecure connections only from the current domain.

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100, 20);
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain:String) {
    domain_txt.text = sendingDomain;
    return (sendingDomain == this.domain() || sendingDomain == "www.macromedia.com");
};
my_lc.allowInsecureDomain = function(sendingDomain:String) {
    return (sendingDomain == this.domain());
}
my_lc.sayHello = function(name:String) {
    welcome_txt.text = "Hello, "+name;
};
my_lc.connect("lc_name");
```

See also

allowDomain (LocalConnection.allowDomain handler), connect (LocalConnection.connect method)

close (LocalConnection.close method)

```
public close() : Void
```

Closes (disconnects) a LocalConnection object. Issue this command when you no longer want the object to accept commands—for example, when you want to issue a `LocalConnection.connect()` command using the same `connectionName` parameter in another SWF file.

Availability

Flash Lite 3.1

See also

`connect (LocalConnection.connect method)`

connect (LocalConnection.connect method)

```
public connect(connectionName:String) : Boolean
```

Prepares a LocalConnection object to receive commands from a `LocalConnection.send()` command (called the *sending LocalConnection object*). The object that calls this command is called the *receiving LocalConnection object*. The receiving and sending objects must be running on the same client computer.

Make sure you define the methods attached to *receiving_lc* before calling this method, as shown in all the examples in this section.

By default, Flash Lite resolves `connectionName` into a value of `"`*superdomain*`:connectionName"`, where *superdomain* is the superdomain of the SWF file containing the `LocalConnection.connect()` command. For example, if the SWF file containing the receiving LocalConnection object is located at www.someDomain.com, `connectionName` resolves to `"someDomain.com:connectionName"`. (If a SWF file is located on the client computer, the value assigned to `superdomain` is `"localhost"`.)

Also by default, Flash Lite lets the receiving LocalConnection object accept commands only from sending LocalConnection objects whose connection name also resolves into a value of `"`*superdomain*`:connectionName"`. In this way, Flash Lite makes it simple for SWF files located in the same domain to communicate with each other.

If you are implementing communication only between SWF files in the same domain, specify a string for `connectionName` that does not begin with an underscore (_) and that does not specify a domain name (for example, `"myDomain:connectionName"`). Use the same string in the `LocalConnection.connect(connectionName)` command.

If you are implementing communication between SWF files in different domains, specifying a string for *connectionName* that begins with an underscore (_) will make the SWF with the receiving LocalConnection object more portable between domains. Here are the two possible cases:

- If the string for *connectionName* does not begin with an underscore (_), Flash Lite adds a prefix with the superdomain and a colon (for example, `"myDomain:connectionName"`). Although this ensures that your connection does not conflict with connections of the same name from other domains, any sending LocalConnection objects must specify this superdomain (for example, `"myDomain:connectionName"`). If the SWF with the receiving LocalConnection object is moved to another domain, the player changes the prefix to reflect the new superdomain (for example, `"anotherDomain:connectionName"`). All sending LocalConnection objects would have to be manually edited to point to the new superdomain.
- If the string for *connectionName* begins with an underscore (for example, `"_connectionName"`), Flash Lite does not add a prefix to the string. This means that the receiving and sending LocalConnection objects will use identical strings for `connectionName`. If the receiving object uses `LocalConnection.allowDomain` to specify that connections from any domain will be accepted, the SWF with the receiving LocalConnection object can be moved to another domain without altering any sending LocalConnection objects.

For more information, see the discussion of `connectionName` in `LocalConnection.send()` and also the `LocalConnection.allowDomain` and `LocalConnection.domain()` entries.

Note: *Colons are used as special characters to separate the superdomain from the `connectionName` string. A string for `connectionName` that contains a colon is not valid.*

Availability

Flash Lite 3.1

Parameters

connectionName : String - A string that corresponds to the connection name specified in the `LocalConnection.send()` command that wants to communicate with *receiving_lc*.

Returns

Boolean - A Boolean value: `true` if no other process running on the same client computer has already issued this command using the same value for the *connectionName* parameter; `false` otherwise.

Example

The following example shows how a SWF file in a particular domain can invoke a method named `printOut` in a receiving SWF file in the same domain.

First, create one SWF file with the following code:

```
this.createTextField("tf", this.getNextHighestDepth(), 10, 10, 300, 100);
var aLocalConnection:LocalConnection = new LocalConnection();
aLocalConnection.connect("demoConnection");
aLocalConnection.printOut = function(aString:String):Void{
    tf.text += aString;
}
```

Then create a second with the following code:

```
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("demoConnection", "printOut", "This is a message from file B. Hello.");
```

To test this example, run the first SWF file, and then run the second one.

See also

send (LocalConnection.send method), allowDomain (LocalConnection.allowDomain handler), domain (LocalConnection.domain method)

domain (LocalConnection.domain method)

```
public domain() : String
```

Returns a string representing the domain of the location of the current SWF file.

In SWF files published for Flash Player 6 or earlier (i.e. Flash Lite 1.x), the returned string is the superdomain of the current SWF file. For example, if the SWF file is located at www.adobe.com, this command returns `"adobe.com"`.

In SWF files published for Flash Player 7 or later (i.e. Flash Lite 2.x and 3.x), the returned string is the exact domain of the current SWF file. For example, if the SWF file is located at www.adobe.com, this command returns `"www.adobe.com"`.

If the current SWF file is a local file residing on the client computer, this command returns `"localhost"`.

The most common way to use this command is to include the domain name of the sending LocalConnection object as a parameter to the method you plan to invoke in the receiving LocalConnection object or with `LocalConnection.allowDomain` to accept commands from a specified domain. If you are enabling communication only between LocalConnection objects that are located in the same domain, you probably don't need to use this command.

Availability

Flash Lite 3.1

Returns

String - A string representing the domain of the location of the current SWF file; for more information, see the Description section.

Example

In the following example, a receiving SWF file accepts commands only from SWF files located in the same domain or at example.com:

```
// If both the sending and receiving SWF files are Flash Player 6,
// then use the superdomain
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain):String{
    return (sendingDomain==this.domain() || sendingDomain=="example.com");
}

// If either the sending or receiving SWF file is Flash Player 7 or later,
// then use the exact domain. In this case, commands from a SWF file posted
// at www.example.com will be accepted, but those from one posted at
// a different subdomain, e.g. test.example.com, will not.
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain):String{
    return (sendingDomain==this.domain() || sendingDomain=="www.example.com");
}
```

In the following example, a sending SWF file located at www.yourdomain.com invokes a method in a receiving SWF file located at www.mydomain.com. The sending SWF file includes its domain name as a parameter to the method it invokes, so the receiving SWF file can return a reply value to a LocalConnection object in the correct domain. The sending SWF file also specifies that it will accept commands only from SWF files at mydomain.com.

Line numbers are included for reference purposes. The sequence of events is described in the following list:

- The receiving SWF file prepares to receive commands on a connection named `"sum"` (line 11). The Flash Lite player resolves the name of this connection to `"mydomain.com:sum"` (see `LocalConnection.connect()`).
- The sending SWF file prepares to receive a reply on the LocalConnection object named `"result"` (line 67). It also specifies that it will accept commands only from SWF files at mydomain.com (lines 51 to 53).
- The sending SWF file invokes the `aSum` method of a connection named `"mydomain.com:sum"` (line 68) and passes the following parameters: its superdomain, the name of the connection to receive the reply (`"result"`), and the values to be used by `aSum` (123 and 456).
- The `aSum` method (line 6) is invoked with the following values: sender = `"mydomain.com:result"`, `replyMethod` = `"aResult"`, `n1` = 123, and `n2` = 456. It then executes the following line of code:

  ```
  this.send("mydomain.com:result", "aResult", (123 + 456));
  ```

- The `aResult` method (line 54) shows the value returned by `aSum` (579).

```
// The receiving SWF at http://www.mydomain.com/folder/movie.swf
// contains the following code

1 var aLocalConnection:LocalConnection = new LocalConnection();
2 aLocalConnection.allowDomain = function()
3 {
    // Allow connections from any domain
4 return true;
5 }
6 aLocalConnection.aSum = function(sender, replyMethod, n1, n2)
7 {
8 this.send(sender, replyMethod, (n1 + n2));
9 }
10
11 aLocalConnection.connect("sum");

// The sending SWF at http://www.yourdomain.com/folder/movie.swf
// contains the following code

50 var lc:LocalConnection = new LocalConnection();
51 lc.allowDomain = function(aDomain) {
       // Allow connections only from mydomain.com
52 return (aDomain == "mydomain.com");
53 }
54 lc.aResult = function(aParam) {
55 trace("The sum is " + aParam);
56 }
    // determine our domain and see if we need to truncate it
57 var channelDomain:String = lc.domain();
58 if (getVersion() >= 7 && this.getSWFVersion() >= 7)
59 {
       // split domain name into elements
60 var domainArray:Array = channelDomain.split(".");

       // if more than two elements are found,
       // chop off first element to create superdomain
61 if (domainArray.length > 2)
62 {
63 domainArray.shift();
64 channelDomain = domainArray.join(".");
65 }
66 }

67 lc.connect("result");
68 lc.send("mydomain.com:sum", "aSum", channelDomain + ':' + "result",
"aResult", 123, 456);
```

See also

allowDomain (LocalConnection.allowDomain handler), connect (LocalConnection.connect method)

LocalConnection constructor

`public LocalConnection()`

Creates a LocalConnection object.

Availability

Flash Lite 3.1

Example

The following example shows how receiving and sending SWF files create LocalConnnection objects. The two SWF files can use the same name or different names for their respective LocalConnection objects. In this example they use different names.

```
// Code in the receiving SWF file
this.createTextField("result_txt", 1, 10, 10, 100, 22);
result_txt.border = true;
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.methodToExecute = function(param1:Number, param2:Number) {
    result_txt.text = param1+param2;
};
receiving_lc.connect("lc_name");
```

The following SWF file sends the request to the first SWF file.

```
// Code in the sending SWF file
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", 5, 7);
```

See also

`connect (LocalConnection.connect method)`, `send (LocalConnection.send method)`

onStatus (LocalConnection.onStatus handler)

```
onStatus = function(infoObject:Object) {}
```

Invoked after a sending LocalConnection object tries to send a command to a receiving LocalConnection object. If you want to respond to this event handler, you must create a function to process the information object sent by the LocalConnection object.

If the information object returned by this event handler contains a level value of status, Flash successfully sent the command to a receiving LocalConnection object. This does not mean that Flash successfully invoked the specified method of the receiving LocalConnection object; it means only that Flash could send the command. For example, the method is not invoked if the receiving LocalConnection object doesn't allow connections from the sending domain or if the method does not exist. The only way to know for sure if the method was invoked is to have the receiving object send a reply to the sending object.

If the information object returned by this event handler contains a level value of error, Flash cannot send the command to a receiving LocalConnection object, most likely because there is no receiving LocalConnection object connected whose name corresponds to the name specified in the `sending_lc.send()` command that invoked this handler.

In addition to this `onStatus` handler, Flash also provides a "super" function called `System.onStatus`. If `onStatus` is invoked for a particular object and there is no function assigned to respond to it, Flash processes a function assigned to `System.onStatus` if it exists.

In most cases, you implement this handler only to respond to error conditions, as shown in the following example.

Availability

Flash Lite 3.1

Parameters

infoObject : Object - A parameter defined according to the status message. For details about this parameter, see the Description section.

Example

The following example displays a status message about whether the SWF file connects to another local connection object called `lc_name`. A TextInput component called `name_ti`, a TextArea instance called `status_ta` and a Button instance called `send_button` are used to display content.

```
var sending_lc:LocalConnection;
var sendListener:Object = new Object();
sendListener.click = function(evt:Object) {
    sending_lc = new LocalConnection();
    sending_lc.onStatus = function(infoObject:Object) {
        switch (infoObject.level) {
        case 'status' :
            status_ta.text = "LocalConnection connected successfully.";
                break;
        case 'error' :
            status_ta.text = "LocalConnection encountered an error.";
            break;
        }
    };
    sending_lc.send("lc_name", "sayHello", name_ti.text);
};
send_button.addEventListener("click", sendListener);
```

See also

send (LocalConnection.send method), onStatus (System.onStatus handler)

send (LocalConnection.send method)

```
public send(connectionName:String, methodName:String, [args:Object]) : Boolean
```

Invokes the method named `method` on a connection opened by a receiving LocalConnection object. The object that calls this method is the sending LocalConnection object. The SWF files that contain the sending and receiving objects must be running on the same client device.

There is a 40 kilobyte limit to the amount of data you can pass as parameters to this command. If the command returns `false` but your syntax is correct, try dividing the `LocalConnection.send()` requests into multiple commands, each with less than 40K of data.

As discussed in the entry `LocalConnection.connect()`, Flash Lite adds the current superdomain to `connectionName` by default. If you are implementing communication between different domains, you need to define `connectionName` in both the sending and receiving LocalConnection objects in such a way that Flash does not add the current superdomain to `connectionName`. You can do this in one of the following two ways:

- Use an underscore (_) at the beginning of `connectionName` in both the sending and receiving LocalConnection objects. In the SWF file containing the receiving object, use `LocalConnection.allowDomain` to specify that connections from any domain will be accepted. This implementation lets you store your sending and receiving SWF files in any domain.

- Include the superdomain in `connectionName` in the sending LocalConnection object—for example, `myDomain.com:myConnectionName`. In the receiving object, use `LocalConnection.allowDomain` to specify that connections from the specified superdomain will be accepted (in this case, myDomain.com) or that connections from any domain will be accepted.

Note: *You cannot specify a superdomain in connectionName in the receiving LocalConnection object—you can only do this in the sending LocalConnection object.*

When using this method, consider the Flash Lite security model. By default, a LocalConnection object is associated with the sandbox of the SWF file that created it, and cross-domain calls to LocalConnection objects are not allowed unless the `LocalConnection.allowDomain()` method has been invoked.

For more information, see the following:

- Chapter 17, "Understanding Security," in *Learning ActionScript 2.0 in Flash*
- The Flash Player 8 Security white paper at http://www.macromedia.com/go/fp8_security
- The Flash Player 8 Security-Related API white paper at http://www.macromedia.com/go/fp8_security_apis

Availability

Flash Lite 3.1

Parameters

connectionName : String - A string that corresponds to the connection name specified in the `LocalConnection.connect()` command that wants to communicate with *sending_lc*.

methodName : String - A string specifying the name of the method to be invoked in the receiving LocalConnection object. The following method names cause the command to fail: `send`, `connect`, `close`, `domain`, `onStatus`, and `allowDomain`.

args : Object [optional] - Arguments to be passed to the specified method.

Returns

Boolean - A Boolean value: `true` if Flash can carry out the request; `false` otherwise.

Note: *A return value of true does not necessarily mean that Flash Lite successfully connected to a receiving LocalConnection object. It means only that the command is syntactically correct. To determine whether the connection succeeded, see LocalConnection.onStatus.*

Example

For an example of communicating between LocalConnection objects located in the same domain, see `LocalConnection.connect()`. For an example of communicating between LocalConnection objects located in specified domains, see `LocalConnection.domain()`.

See also

`allowDomain (LocalConnection.allowDomain handler)`, `connect (LocalConnection.connect method)`, `domain (LocalConnection.domain method)`, `onStatus (LocalConnection.onStatus handler)`

Math

```
Object
    |
    +-Math
```

```
public class Math
extends Object
```

The Math class is a top-level class whose methods and properties you can use without using a constructor.

Use the methods and properties of this class to access and manipulate mathematical constants and functions. All the properties and methods of the Math class are static and must be called using the syntax `Math.method()` or `Math.CONSTANT`. In ActionScript, constants are defined with the maximum precision of double-precision IEEE-754 floating-point numbers.

Several Math class methods use the measure of an angle in radians as a parameter. You can use the following equation to calculate radian values before calling the method and then provide the calculated value as the parameter, or you can provide the entire right side of the equation (with the angle's measure in degrees in place of `degrees`) as the radian parameter.

To calculate a radian value, use the following formula:

```
radians = degrees * Math.PI/180
```

The following is an example of passing the equation as a parameter to calculate the sine of a 45° angle:

`Math.sin(45 * Math.PI/180)` is the same as `Math.sin(.7854)`

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
`static`	E : Number	A mathematical constant for the base of natural logarithms, expressed as *e*.
`static`	LN10 : Number	A mathematical constant for the natural logarithm of 10, expressed as loge10, with an approximate value of 2.302585092994046.
`static`	LN2 : Number	A mathematical constant for the natural logarithm of 2, expressed as loge2, with an approximate value of 0.6931471805599453.
`static`	LOG10E : Number	A mathematical constant for the base-10 logarithm of the constant e (`Math.E`), expressed as log10e, with an approximate value of 0.4342944819032518.
`static`	LOG2E : Number	A mathematical constant for the base-2 logarithm of the constant e (`Math.E`), expressed as log2e, with an approximate value of 1.442695040888963387.

Modifiers	Property	Description
static	PI : Number	A mathematical constant for the ratio of the circumference of a circle to its diameter, expressed as pi, with a value of 3.141592653589793.
static	SQRT1_2 : Number	A mathematical constant for the square root of one-half, with an approximate value of 0.7071067811865476.
static	SQRT2 : Number	A mathematical constant for the square root of 2, with an approximate value of 1.4142135623730951.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property). prototype (Object.prototype property), __resolve (Object.__resolve property)

Method summary

Modifiers	Signature	Description
static	abs (x : Number) : Number	Computes and returns an absolute value for the number specified by the parameter x.
static	acos (x : Number) : Number	Computes and returns the arc cosine of the number specified in the parameter x, in radians.
static	asin (x : Number) : Number	Computes and returns the arc sine for the number specified in the parameter x, in radians.
static	atan (tangent : Number) : Number	Computes and returns the value, in radians, of the angle whose tangent is specified in the parameter tangent.
static	atan2 (y : Number, x : Number) : Number	Computes and returns the angle of the point y/x in radians, when measured counterclockwise from a circle's *x* axis (where 0,0 represents the center of the circle).
static	ceil (x : Number) : Number	Returns the ceiling of the specified number or expression.
static	cos (x : Number) : Number	Computes and returns the cosine of the specified angle in radians.
static	exp (x : Number) : Number	Returns the value of the base of the natural logarithm (*e*), to the power of the exponent specified in the parameter x.
static	floor (x : Number) : Number	Returns the floor of the number or expression specified in the parameter x.
static	log (x : Number) : Number	Returns the natural logarithm of parameter x.
static	max (x : Number, y : Number) : Number	Evaluates x and y and returns the larger value.
static	min (x : Number, y : Number) : Number	Evaluates x and y and returns the smaller value.
static	pow (x : Number, y : Number) : Number	Computes and returns x to the power of y.
static	random () : Number	Returns a pseudo-random number n, where 0 <= n < 1.
static	round (x : Number) : Number	Rounds the value of the parameter x up or down to the nearest integer and returns the value.

Modifiers	Signature	Description
static	sin(x:Number) : Number	Computes and returns the sine of the specified angle in radians.
static	sqrt(x:Number) : Number	Computes and returns the square root of the specified number.
static	tan(x:Number) : Number	Computes and returns the tangent of the specified angle.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

abs (Math.abs method)

```
public static abs(x:Number) : Number
```

Computes and returns an absolute value for the number specified by the parameter x.

Availability

Flash Lite 2.0

Parameters

x: Number - A number.

Returns

Number - A number.

Example

The following example shows how `Math.abs()` returns the absolute value of a number and does not affect the value of the x parameter (called `num` in this example):

```
var num:Number = -12;
var numAbsolute:Number = Math.abs(num);
trace(num); // output: -12
trace(numAbsolute); // output: 12
```

acos (Math.acos method)

```
public static acos(x:Number) : Number
```

Computes and returns the arc cosine of the number specified in the parameter x, in radians.

Availability

Flash Lite 2.0

Parameters

x: Number - A number from -1.0 to 1.0.

Returns

Number - A number; the arc cosine of the parameter x.

Example

The following example displays the arc cosine for several values.

```
trace(Math.acos(-1)); // output: 3.14159265358979
trace(Math.acos(0)); // output: 1.5707963267949
trace(Math.acos(1)); // output: 0
```

See also

asin (Math.asin method), atan (Math.atan method), atan2 (Math.atan2 method), cos (Math.cos method), sin (Math.sin method), tan (Math.tan method)

asin (Math.asin method)

```
public static asin(x:Number) : Number
```

Computes and returns the arc sine for the number specified in the parameter x, in radians.

Availability

Flash Lite 2.0

Parameters

x: Number - A number from -1.0 to 1.0.

Returns

Number - A number between negative pi divided by 2 and positive pi divided by 2.

Example

The following example displays the arc sine for several values.

```
trace(Math.asin(-1)); // output: -1.5707963267949
trace(Math.asin(0)); // output: 0
trace(Math.asin(1)); // output: 1.5707963267949
```

See also

acos (Math.acos method), atan (Math.atan method), atan2 (Math.atan2 method), cos (Math.cos method), sin (Math.sin method), tan (Math.tan method)

atan (Math.atan method)

```
public static atan(tangent:Number) : Number
```

Computes and returns the value, in radians, of the angle whose tangent is specified in the parameter `tangent`. The return value is between negative pi divided by 2 and positive pi divided by 2.

Availability

Flash Lite 2.0

Parameters

tangent: Number - A number that represents the tangent of an angle.

Returns

Number - A number between negative pi divided by 2 and positive pi divided by 2.

Example

The following example displays the angle value for several tangents.

```
trace(Math.atan(-1)); // output: -0.785398163397448
trace(Math.atan(0)); // output: 0
trace(Math.atan(1)); // output: 0.785398163397448
```

See also

acos (Math.acos method), asin (Math.asin method), atan2 (Math.atan2 method), cos (Math.cos method), sin (Math.sin method), tan (Math.tan method)

atan2 (Math.atan2 method)

```
public static atan2(y:Number, x:Number) : Number
```

Computes and returns the angle of the point y/x in radians, when measured counterclockwise from a circle's *x* axis (where 0,0 represents the center of the circle). The return value is between positive pi and negative pi.

Availability

Flash Lite 2.0

Parameters

y: Number - A number specifying the *y* coordinate of the point.

x: Number - A number specifying the *x* coordinate of the point.

Returns

Number - A number.

Example

The following example returns the angle, in radians, of the point specified by the coordinates (0, 10), such that x = 0 and y = 10. Note that the first parameter to atan2 is always the y coordinate.

```
trace(Math.atan2(10, 0)); // output: 1.5707963267949
```

See also

acos (Math.acos method), asin (Math.asin method), atan (Math.atan method), cos (Math.cos method), sin (Math.sin method), tan (Math.tan method)

ceil (Math.ceil method)

```
public static ceil(x:Number) : Number
```

Returns the ceiling of the specified number or expression. The ceiling of a number is the closest integer that is greater than or equal to the number.

Availability
Flash Lite 2.0

Parameters
x: Number - A number or expression.

Returns
Number - An integer that is both closest to, and greater than or equal to, parameter x.

Example
The following code returns a value of 13:

```
Math.ceil(12.5);
```

See also
floor (Math.floor method), round (Math.round method)

cos (Math.cos method)

```
public static cos(x:Number) : Number
```

Computes and returns the cosine of the specified angle in radians. To calculate a radian, see the description of the Math class entry.

Availability
Flash Lite 2.0

Parameters
x: Number - A number that represents an angle measured in radians.

Returns
Number - A number from -1.0 to 1.0.

Example
The following example displays the cosine for several different angles.

```
trace (Math.cos(0)); // 0 degree angle. Output: 1
trace (Math.cos(Math.PI/2)); // 90 degree angle. Output: 6.12303176911189e-17
trace (Math.cos(Math.PI)); // 180 degree angle. Output: -1
trace (Math.cos(Math.PI*2)); // 360 degree angle. Output: 1
```

Note: The cosine of a 90 degree angle is zero, but because of the inherent inaccuracy of decimal calculations using binary numbers, Flash Lite player will report a number extremely close to, but not exactly equal to, zero.

See also
acos (Math.acos method), asin (Math.asin method), atan (Math.atan method), atan2 (Math.atan2 method), sin (Math.sin method), tan (Math.tan method)

E (Math.E property)

```
public static E : Number
```

A mathematical constant for the base of natural logarithms, expressed as *e*. The approximate value of *e* is 2.71828182845905.

Availability
Flash Lite 2.0

Example
This example shows how `Math.E` is used to compute continuously compounded interest for a simple case of 100 percent interest over a one-year period.

```
var principal:Number = 100;
var simpleInterest:Number = 100;
var continuouslyCompoundedInterest:Number = (100 * Math.E) - principal;

trace ("Beginning principal: $" + principal);
trace ("Simple interest after one year: $" + simpleInterest);
trace ("Continuously compounded interest after one year: $" + continuouslyCompoundedInterest);

//
Output:
Beginning principal: $100
Simple interest after one year: $100
Continuously compounded interest after one year: $171.828182845905
```

exp (Math.exp method)

```
public static exp(x:Number) : Number
```

Returns the value of the base of the natural logarithm (*e*), to the power of the exponent specified in the parameter `x`. The constant `Math.E` can provide the value of *e*.

Availability
Flash Lite 2.0

Parameters
x: Number - The exponent; a number or expression.

Returns
Number - A number.

Example
The following example displays the logarithm for two number values.

```
trace(Math.exp(1)); // output: 2.71828182845905
trace(Math.exp(2)); // output: 7.38905609893065
```

See also
E (Math.E property)

floor (Math.floor method)

```
public static floor(x:Number) : Number
```

Returns the floor of the number or expression specified in the parameter `x`. The floor is the closest integer that is less than or equal to the specified number or expression.

Availability
Flash Lite 2.0

Parameters
x: Number - A number or expression.

Returns
Number - The integer that is both closest to, and less than or equal to, parameter `x`.

Example
The following code returns a value of 12:

```
Math.floor(12.5);
```

The following code returns a value of -7:

```
Math.floor(-6.5);
```

LN10 (Math.LN10 property)

```
public static LN10 : Number
```

A mathematical constant for the natural logarithm of 10, expressed as loge10, with an approximate value of 2.302585092994046.

Availability
Flash Lite 2.0

Example
This example traces the value of `Math.LN10`.

```
trace(Math.LN10);
// output: 2.30258509299405
```

LN2 (Math.LN2 property)

```
public static LN2 : Number
```

A mathematical constant for the natural logarithm of 2, expressed as loge2, with an approximate value of 0.6931471805599453.

Availability
Flash Lite 2.0

log (Math.log method)

```
public static log(x:Number) : Number
```

Returns the natural logarithm of parameter `x`.

Availability
Flash Lite 2.0

Parameters
x: Number - A number or expression with a value greater than 0.

Returns
Number - The natural logarithm of parameter `x`.

Example
The following example displays the logarithm for three numerical values.

```
trace(Math.log(0)); // output: -Infinity
trace(Math.log(1)); // output: 0
trace(Math.log(2)); // output: 0.693147180559945
trace(Math.log(Math.E)); // output: 1
```

LOG10E (Math.LOG10E property)

```
public static LOG10E : Number
```

A mathematical constant for the base-10 logarithm of the constant e (`Math.E`), expressed as log10e, with an approximate value of 0.4342944819032518.

The `Math.log()` method computes the natural logarithm of a number. Multiply the result of `Math.log()` by `Math.LOG10E` obtain the base-10 logarithm.

Availability
Flash Lite 2.0

Example
This example shows how to obtain the base-10 logarithm of a number:

```
trace(Math.log(1000) * Math.LOG10E);
// Output: 3
```

LOG2E (Math.LOG2E property)

```
public static LOG2E : Number
```

A mathematical constant for the base-2 logarithm of the constant e (`Math.E`), expressed as log2e, with an approximate value of 1.442695040888963387.

The `Math.log` method computes the natural logarithm of a number. Multiply the result of `Math.log()` by `Math.LOG2E` obtain the base-2 logarithm.

Availability
Flash Lite 2.0

Example
This example shows how to obtain the base-2 logarithm of a number:

```
trace(Math.log(16) * Math.LOG2E);
// Output: 4
```

max (Math.max method)

```
public static max(x:Number, y:Number) : Number
```

Evaluates `x` and `y` and returns the larger value.

Availability
Flash Lite 2.0

Parameters
x: Number - A number or expression.

y: Number - A number or expression.

Returns
Number - A number.

Example
The following example displays `Thu Dec 30 00:00:00 GMT-0700 2004`, which is the larger of the evaluated expressions.

```
var date1:Date = new Date(2004, 11, 25);
var date2:Date = new Date(2004, 11, 30);
var maxDate:Number = Math.max(date1.getTime(), date2.getTime());
trace(new Date(maxDate).toString());
```

See also
min (Math.min method)

min (Math.min method)

```
public static min(x:Number, y:Number) : Number
```

Evaluates `x` and `y` and returns the smaller value.

Availability
Flash Lite 2.0

Parameters
x: Number - A number or expression.

y: Number - A number or expression.

Returns
Number - A number.

Example

The following example displays `Sat Dec 25 00:00:00 GMT-0700 2004`, which is the smaller of the evaluated expressions.

```
var date1:Date = new Date(2004, 11, 25);
var date2:Date = new Date(2004, 11, 30);
var minDate:Number = Math.min(date1.getTime(), date2.getTime());
trace(new Date(minDate).toString());
```

See also

max (Math.max method)

PI (Math.PI property)

```
public static PI : Number
```

A mathematical constant for the ratio of the circumference of a circle to its diameter, expressed as pi, with a value of 3.141592653589793.

Availability

Flash Lite 2.0

Example

The following example draws a circle using the mathematical constant pi and the Drawing API.

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x, Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

pow (Math.pow method)

```
public static pow(x:Number, y:Number) : Number
```

Computes and returns `x` to the power of `y`.

Availability

Flash Lite 2.0

Parameters

x: Number - A number to be raised to a power.

y: Number - A number specifying a power the parameter x is raised to.

Returns
Number - A number.

random (Math.random method)

```
public static random() : Number
```

Returns a pseudo-random number n, where 0 <= n < 1. The number returned is a pseudo-random number because it is not generated by a truly random natural phenomenon such as radioactive decay.

Availability
Flash Lite 2.0

Returns
Number - A number.

Example
The following example outputs 100 random integers between 4 and 11 (inclusively):

```
function randRange(min:Number, max:Number):Number {
    var randomNum:Number = Math.floor(Math.random() * (max - min + 1)) + min;
    return randomNum;
}
for (var i = 0; i < 100; i++) {
    var n:Number = randRange(4, 11)
    trace(n);
}
```

round (Math.round method)

```
public static round(x:Number) : Number
```

Rounds the value of the parameter x up or down to the nearest integer and returns the value. If parameter x is equidistant from its two nearest integers (that is, the number ends in .5), the value is rounded up to the next higher integer.

Availability
Flash Lite 2.0

Parameters
x: Number - A number.

Returns
Number - A number; an integer.

Example
The following example returns a random number between two specified integers.

```
function randRange(min:Number, max:Number):Number {
    var randomNum:Number = Math.round(Math.random() * (max-min+1) + (min-.5));
    return randomNum;
}
for (var i = 0; i<25; i++) {
    trace(randRange(4, 11));
}
```

See also

ceil (Math.ceil method), floor (Math.floor method)

sin (Math.sin method)

```
public static sin(x:Number) : Number
```

Computes and returns the sine of the specified angle in radians. To calculate a radian, see the description of the Math class entry.

Availability

Flash Lite 2.0

Parameters

x: Number - A number that represents an angle measured in radians.

Returns

Number - A number; the sine of the specified angle (between -1.0 and 1.0).

Example

The following example draws a circle using the mathematical constant pi, the sine of an angle, and the Drawing API.

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x, Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

See also

acos (Math.acos method), asin (Math.asin method), atan (Math.atan method), atan2 (Math.atan2 method), cos (Math.cos method), tan (Math.tan method)

sqrt (Math.sqrt method)

```
public static sqrt(x:Number) : Number
```

Computes and returns the square root of the specified number.

Availability

Flash Lite 2.0

Parameters

x: Number - A number or expression greater than or equal to 0.

Returns

Number - A number if parameter *x* is greater than or equal to zero; NaN (not a number) otherwise.

SQRT1_2 (Math.SQRT1_2 property)

```
public static SQRT1_2 : Number
```

A mathematical constant for the square root of one-half, with an approximate value of 0.7071067811865476.

Availability

Flash Lite 2.0

Example

This example traces the value of `Math.SQRT1_2`.

```
trace(Math.SQRT1_2);
// Output: 0.707106781186548
```

SQRT2 (Math.SQRT2 property)

```
public static SQRT2 : Number
```

A mathematical constant for the square root of 2, with an approximate value of 1.4142135623730951.

Availability

Flash Lite 2.0

Example

This example traces the value of `Math.SQRT2`.

```
trace(Math.SQRT2);
// Output: 1.4142135623731
```

tan (Math.tan method)

```
public static tan(x:Number) : Number
```

Computes and returns the tangent of the specified angle. To calculate a radian, use the information outlined in the introduction to the Math class.

Availability

Flash Lite 2.0

Parameters

x: Number - A number that represents an angle measured in radians.

Returns

Number - A number; tangent of parameter x.

Example

The following example draws a circle using the mathematical constant pi, the tangent of an angle, and the Drawing API.

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x, Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

See also

acos (Math.acos method), asin (Math.asin method), atan (Math.atan method), atan2 (Math.atan2 method), cos (Math.cos method), sin (Math.sin method)

Matrix (flash.geom.Matrix)

```
Object
    |
    +-flash.geom.Matrix

public class Matrix
extends Object
```

The flash.geom.Matrix class represents a transformation matrix that determines how to map points from one coordinate space to another. By setting the properties of a Matrix object and applying it to a MovieClip or BitmapData object, you can perform various graphical transformations on the object. These transformation functions include translation (*x* and *y* repositioning), rotation, scaling, and skewing.

Together these types of transformations are known as *affine transformations*. Affine transformations preserve the straightness of lines during transformations, and parallel lines stay parallel.

To apply a transformation matrix to a movie clip, create a flash.geom.Transform the object, and set its Matrix property to the transformation matrix. Matrix objects are also used as parameters of some methods, such as the `draw()` method of the flash.display.BitmapData class.

A transformation matrix object is considered a 3 x 3 matrix with the following contents:

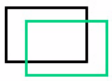
$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ u & v & w \end{bmatrix}$$

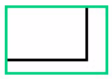
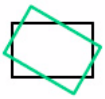
In traditional transformation matrixes, the `u`, `v`, and `w` properties provide extra capabilities. The Matrix class can only operate in two-dimensional space so it always assumes that the property values `u` and `v` are 0.0, and that the property value `w` is 1.0. In other words the effective values of the matrix are as follows:

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

You can get and set the values of all six of the other properties in a Matrix object: `a`, `b`, `c`, `d`, `tx`, and `ty`.

The Matrix class supports the four major types of transformation functions: translation, scaling, rotation, and skewing. There are specialized methods for three of these functions, as described in the following table.

Transformation	Method	Matrix values	Display result	Description
Translation (displacement)	`translate(tx, ty)`	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Moves the image `tx` pixels to the right and `ty` pixels down.

Transformation	Method	Matrix values	Display result	Description
Scaling	`scale(sx, sy)`	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$		Resizes the image, multiplying the location of each pixel by `sx` on the *x* axis and `sy` on the *y* axis.
Rotation	`rotate(q)`	$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$		Rotates the image by an angle `q`, which is measured in radians
Skewing or shearing	None; must set the properties `b` and `c`.	$\begin{bmatrix} 0 & sk_y & 0 \\ sk_x & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		Progressively slides the image in a direction parallel to the *x* or *y* axis. The value `skx` acts as a multiplier controlling the sliding distance along the *x* axis; `sky` controls the sliding distance along the *y* axis.

Each transformation function alters the current matrix properties so that you can effectively combine multiple transformations. To do this, you call more than one transformation function before applying the matrix to its movie clip or bitmap target.

Availability

Flash Lite 3.1

See also

`transform (MovieClip.transform property)`, `Transform (flash.geom.Transform)`, `draw (BitmapData.draw method)`, `a (Matrix.a property)`, `b (Matrix.b property)`, `c (Matrix.c property)`, `d (Matrix.d property)`, `tx (Matrix.tx property)`, `ty (Matrix.ty property)`, `translate (Matrix.translate method)`, `scale (Matrix.scale method)`, `rotate (Matrix.rotate method)`

Property summary

Modifiers	Property	Description
	a:Number	The value in the first row and first column of the Matrix object, which affects the positioning of pixels along the *x* axis when scaling or rotating an image.
	b:Number	The value in the first row and second column of the Matrix object, which affects the positioning of pixels along the *y* axis when rotating or skewing an image.
	c :Number	The value in the second row and first column of the Matrix object, which affects the positioning of pixels along the *x* axis when rotating or skewing an image.

Modifiers	Property	Description
	d:Number	The value in the second row and second column of the Matrix object, which affects the positioning of pixels along the *y* axis when scaling or rotating an image.
	tx:Number	The distance by which to translate each point along the *x* axis.
	ty:Number	The distance by which to translate each point along the *y* axis.

```
constructor (Object.constructor property), __proto__ (Object.__proto__ property),
prototype (Object.prototype property), __resolve (Object.__resolve property)
```

Constructor summary

Signature	Description
Matrix([a:Number], [b:Number], [c:Number], [d:Number], [tx:Number], [ty:Number])	Creates a new Matrix object with the specified parameters.

Method summary

Modifiers	Signature	Description
	clone(): Matrix	Returns a new Matrix object that is a clone of this matrix, with an exact copy of the contained object.
	concat(m:Matrix) : Void	Concatenates a matrix with the current matrix, effectively combining the geometric effects of the two.
	createBox(scaleX:Number, scaleY:Number, [rotation:Number], [tx:Number], [ty:Number]) : Void	Includes parameters for scaling, rotation, and translation.
	createGradientBox(width:Number, height:Number, [rotation:Number], [tx:Number], [ty:Number:]) : Void	Creates the specific style of matrix expected by the MovieClip.beginGradientFill() method.
	deltaTransformPoint(pt:Point) :Point	Given a point in the pretransform coordinate space, returns the coordinates of that point after the transformation occurs.
	identity() : Void	Sets each matrix property to a value that cause a transformed movie clip or geometric construct to be identical to the original.
	invert() : Void	Performs the opposite transformation of the original matrix.
	rotate(angle:Number) : Void	Sets the values in the current matrix so that the matrix can be used to apply a rotation transformation.
	scale(sx:Number, sy:Number) : Void	Modifies a matrix so that its effect, when applied, is to resize an image.

Modifiers	Signature	Description
	toString() : String	Returns a text value listing the properties of the Matrix object.
	transformPoint(pt:Point):Point	Applies the geometric transformation represented by the Matrix object to the specified point.
	translate(tx:Number, ty:Number) : Void	Modifies a Matrix object so that the effect of its transformation is to move an object along the *x* and *y* axes.

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

a (Matrix.a property)

```
public a : Number
```

The value in the first row and first column of the Matrix object, which affects the positioning of pixels along the *x* axis when scaling or rotating an image.

Availability

Flash Lite 3.1

Example

The following example creates the Matrix object `myMatrix` and sets its `a` value.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.a); // 1

myMatrix.a = 2;
trace(myMatrix.a); // 2
```

b (Matrix.b property)

```
public b : Number
```

The value in the first row and second column of the Matrix object, which affects the positioning of pixels along the *y* axis when rotating or skewing an image.

Availability

Flash Lite 3.1

Example

The following example creates the Matrix object `myMatrix` and sets its `b` value.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.b); // 0

var degrees:Number = 45;
var radians:Number = (degrees/180) Math.PI;
myMatrix.b = radians;
trace(myMatrix.b); // 0.785398163397448
```

c (Matrix.c property)

```
public c : Number
```

The value in the second row and first column of the Matrix object, which affects the positioning of pixels along the *x* axis when rotating or skewing an image.

Availability

Flash Lite 3.1

Example

The following example creates the Matrix object `myMatrix` and sets its `c` value.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.c); // 0

var degrees:Number = 45;
var radians:Number = (degrees/180) Math.PI;
myMatrix.c = radians;
trace(myMatrix.c); // 0.785398163397448
```

clone (Matrix.clone method)

```
public clone() : Matrix
```

Returns a new Matrix object that is a clone of this matrix, with an exact copy of the contained object.

Availability

Flash Lite 3.1

Returns

Matrix - A Matrix object.

Example

The following example creates the `clonedMatrix` variable from the `myMatrix` variable. The Matrix class does not have an `equals` method, so the following example uses a custom written function to test the equality of two matrixes.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 0, 0);
var clonedMatrix:Matrix = new Matrix();

trace(myMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(clonedMatrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
trace(equals(myMatrix, clonedMatrix)); // false

clonedMatrix = myMatrix.clone();

trace(myMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(clonedMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(equals(myMatrix, clonedMatrix)); // true

function equals(m1:Matrix, m2:Matrix):Boolean {
    return m1.toString() == m2.toString();
}
```

concat (Matrix.concat method)

```
public concat(m:Matrix) : Void
```

Concatenates a matrix with the current matrix, effectively combining the geometric effects of the two. In mathematical terms, concatenating two matrixes is the same as combining them using matrix multiplication.

For example, if matrix `m1` scales an object by a factor of four, and matrix `m2` rotates an object by 1.5707963267949 radians (`Math.PI/2`), `m1.concat(m2)` transforms `m1` into a matrix that scales an object by a factor of four and rotates the object by `Math.PI/2` radians.

This method replaces the source matrix with the concatenated matrix. If you want to concatenate two matrixes without altering either of the two source matrixes, you can first copy the source matrix the clone() method, as shown in the Example section.

Availability

Flash Lite 3.1

Parameters

m : Matrix - The matrix to be concatenated to the source matrix.

Example

The following example creates three matrixes that define transformations for three rectangle movie clips. The first two matrixes `rotate45Matrix` and `doubleScaleMatrix` are applied to the two rectangles `rectangleMc_1` and `rectangleMc_2`. Then the third matrix is created using the `concat()` method on `rotate45Matrix` and `doubleScaleMatrix` to create `scaleAndRotateMatrix`. This matrix is then applied to `rectangleMc_3` to scale and rotate it.

```
import flash.geom.Matrix;
import flash.geom.Transform;

var rectangleMc_0:MovieClip = createRectangle(20, 80, 0x000000);
var rectangleMc_1:MovieClip = createRectangle(20, 80, 0xFF0000);
var rectangleMc_2:MovieClip = createRectangle(20, 80, 0x00FF00);
var rectangleMc_3:MovieClip = createRectangle(20, 80, 0x0000FF);

var rectangleTrans_1:Transform = new Transform(rectangleMc_1);
var rectangleTrans_2:Transform = new Transform(rectangleMc_2);
var rectangleTrans_3:Transform = new Transform(rectangleMc_3);

var rotate45Matrix:Matrix = new Matrix();
rotate45Matrix.rotate(Math.PI/4);
rectangleTrans_1.matrix = rotate45Matrix;
rectangleMc_1._x = 100;
trace(rotate45Matrix.toString()); // (a=0.707106781186548, b=0.707106781186547, c=-
0.707106781186547, d=0.707106781186548, tx=0, ty=0)

var doubleScaleMatrix:Matrix = new Matrix();
doubleScaleMatrix.scale(2, 2);
rectangleTrans_2.matrix = doubleScaleMatrix;
rectangleMc_2._x = 200;
trace(doubleScaleMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

var scaleAndRotateMatrix:Matrix = doubleScaleMatrix.clone();
scaleAndRotateMatrix.concat(rotate45Matrix);
rectangleTrans_3.matrix = scaleAndRotateMatrix;
rectangleMc_3._x = 300;
trace(scaleAndRotateMatrix.toString()); // (a=1.4142135623731, b=1.41421356237309, c=-
1.41421356237309, d=1.4142135623731, tx=0, ty=0)

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

createBox (Matrix.createBox method)

`public createBox(scaleX:Number, scaleY:Number, [rotation:Number], [tx:Number], [ty:Number]) : Void`

Includes parameters for scaling, rotation, and translation. When applied to a matrix, it sets the matrix's values based on those parameters.

Using the `createBox()` method lets you obtain the same matrix as you would if you were to apply the `identity()`, `rotate()`, `scale()`, and `translate()` methods in succession. For example, `mat1.createBox(2,2,Math.PI/5, 100, 100)` has the same effect as the following:

```
import flash.geom.Matrix;

var mat1:Matrix = new Matrix();
mat1.identity();
mat1.rotate(Math.PI/4);
mat1.scale(2,2);
mat1.translate(10,20);
```

Availability
Flash Lite 3.1

Parameters
scaleX : Number - The factor by which to scale horizontally.

scaleY : Number - The factor by which scale vertically.

rotation : Number [optional] - The amount to rotate, in radians. The default value is 0.

tx : Number [optional] - The number of pixels to translate (move) to the right along the *x* axis. The default value is 0.

ty : Number [optional] - The number of pixels to translate (move) down along the *y* axis. The default value is 0.

Example
The following example sets the `scaleX`, `scaleY` scale, rotation, *x* location, and *y* location of `myMatrix` by calling its `createBox()` method.

```
import flash.geom.Matrix;
import flash.geom.Transform;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.createBox(1, 2, Math.PI/4, 100, 200);
trace(myMatrix.toString()); // (a=0.707106781186548, b=1.41421356237309, c=-
0.707106781186547, d=1.4142135623731, tx=100, ty=200)

var rectangleMc:MovieClip = createRectangle(20, 80, 0xFF0000);
var rectangleTrans:Transform = new Transform(rectangleMc);
rectangleTrans.matrix = myMatrix;
```

createGradientBox (Matrix.createGradientBox method)

```
public createGradientBox(width:Number, height:Number, [rotation:Number], [tx:Number],
[ty:Number]) : Void
```

Creates the specific style of matrix expected by the `MovieClip.beginGradientFill()` method. Width and height are scaled to a `scaleX`/`scaleY` pair and the `tx`/`ty` values are offset by half the width and height.

Availability
Flash Lite 3.1

Parameters
width : Number - The width of the gradient box.

height : Number - The height of the gradient box.

rotation : Number [optional] - The amount to rotate, in radians. The default value is 0.

tx : Number [optional] - The distance in pixels to translate to the right along the *x* axis. This value will be offset by half of the width parameter. The default value is 0.

ty : Number [optional] - The distance in pixels to translate down along the *y* axis. This value will be offset by half of the height parameter. The default value is 0.

Example

The following example uses `myMatrix` as a parameter for the MovieClip object's `beginGradientFill()` method.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.createGradientBox(200, 200, 0, 50, 50);
trace(myMatrix.toString()); // (a=0.1220703125, b=0, c=0, d=0.1220703125, tx=150, ty=150)

var depth:Number = this.getNextHighestDepth();
var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
var colors:Array = [0xFF0000, 0x0000FF];
var alphas:Array = [100, 100];
var ratios:Array = [0, 0xFF];
mc.beginGradientFill("linear", colors, alphas, ratios, myMatrix);
mc.lineTo(0, 300);
mc.lineTo(300, 300);
mc.lineTo(300, 0);
mc.lineTo(0, 0);
```

See also

beginGradientFill (MovieClip.beginGradientFill method)

d (Matrix.d property)

```
public d : Number
```

The value in the second row and second column of the Matrix object, which affects the positioning of pixels along the *y* axis when scaling or rotating an image.

Availability

Flash Lite 3.1

Example

The following example creates the Matrix object `myMatrix` and sets its `d` value.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.d); // 1

myMatrix.d = 2;
trace(myMatrix.d); // 2
```

deltaTransformPoint (Matrix.deltaTransformPoint method)

```
public deltaTransformPoint(pt:Point) : Point
```

Given a point in the pretransform coordinate space, returns the coordinates of that point after the transformation occurs. Unlike the standard transformation applied using the transformPoint() method, the deltaTransformPoint() method's transformation does not consider the translation parameters `tx` and `ty`.

Availability

Flash Lite 3.1

Parameters

pt : Point - A Point object.

Returns

Point - The new Point object.

Example

The following example uses the `deltaTransformPoint()` method to create `deltaTransformedPoint` from `myPoint`. In the example, the `translate()` method does not alter the position of the point named `deltaTransformedPoint`. However, the `scale()` method does affect that point's position. It increases the point's `x` value by a factor of three from 50 to 150.

```
import flash.geom.Matrix;
import flash.geom.Point;

var myMatrix:Matrix = new Matrix();
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.translate(100, 0);
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=100, ty=0)

myMatrix.scale(3, 3);
trace(myMatrix); // (a=3, b=0, c=0, d=3, tx=300, ty=0)

var myPoint:Point = new Point(50,0);
trace(myPoint); // (50, 0)

var deltaTransformedPoint:Point = myMatrix.deltaTransformPoint(myPoint);
trace(deltaTransformedPoint); // (150, 0)

var pointMc_0:MovieClip = createRectangle(10, 10, 0xFF0000);
pointMc_0._x = myPoint.x;

var pointMc_1:MovieClip = createRectangle(10, 10, 0x00FF00);
pointMc_1._x = deltaTransformedPoint.x;

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

identity (Matrix.identity method)

```
public identity() : Void
```

Sets each matrix property to a value that causes a transformed movie clip or geometric construct to be identical to the original.

After calling the `identity()` method, the resulting matrix has the following properties: `a=1, b=0, c=0, d=1, tx=0, ty=0`.

In matrix notation, the identity matrix looks like this:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Availability

Flash Lite 3.1

Example

The following example demonstrates that calling the `identity()` method converts the calling Matrix object to an identity Matrix object. The number and types of transformations applied to the original Matrix object beforehand are irrelevant. If `identity()` is called, the Matrix values are converted to (a=1, b=0, c=0, d=1, tx=0, ty=0).

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 0 ,0);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

myMatrix.rotate(Math.atan(3/4));
trace(myMatrix.toString()); // (a=1.6, b=1.2, c=-1.2, d=1.6, tx=0, ty=0)

myMatrix.translate(100,200);
trace(myMatrix.toString()); // (a=1.6, b=1.2, c=-1.2, d=1.6, tx=100, ty=200)

myMatrix.scale(2, 2);
trace(myMatrix.toString()); // (a=3.2, b=2.4, c=-2.4, d=3.2, tx=200, ty=400)

myMatrix.identity();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
```

invert (Matrix.invert method)

```
public invert() : Void
```

Performs the opposite transformation of the original matrix. You can apply an inverted matrix to an object to undo the transformation performed when applying the original matrix.

Availability

Flash Lite 3.1

Example

The following example creates `halfScaleMatrix` by calling the `invert()` method of `doubleScaleMatrix`, and then demonstrates that the two are Matrix inverses of one another, that is, matrixes that undo any transformations performed by the other. The example shows this inversion by creating `originalAndInverseMatrix`, which is equal to `noScaleMatrix`.

```
import flash.geom.Matrix;
import flash.geom.Transform;

var rectangleMc_0:MovieClip = createRectangle(20, 80, 0xFF0000);
var rectangleMc_1:MovieClip = createRectangle(20, 80, 0x00FF00);
var rectangleMc_2:MovieClip = createRectangle(20, 80, 0x0000FF);
var rectangleMc_3:MovieClip = createRectangle(20, 80, 0x000000);

var rectangleTrans_0:Transform = new Transform(rectangleMc_0);
var rectangleTrans_1:Transform = new Transform(rectangleMc_1);
var rectangleTrans_2:Transform = new Transform(rectangleMc_2);
var rectangleTrans_3:Transform = new Transform(rectangleMc_3);

var doubleScaleMatrix:Matrix = new Matrix(2, 0, 0, 2, 0, 0);
rectangleTrans_0.matrix = doubleScaleMatrix;
trace(doubleScaleMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

var noScaleMatrix:Matrix = new Matrix(1, 0, 0, 1, 0, 0);
rectangleTrans_1.matrix = noScaleMatrix;
rectangleMc_1._x = 100;
trace(noScaleMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var halfScaleMatrix:Matrix = doubleScaleMatrix.clone();
halfScaleMatrix.invert();
rectangleTrans_2.matrix = halfScaleMatrix;
rectangleMc_2._x = 200;
trace(halfScaleMatrix.toString()); // (a=0.5, b=0, c=0, d=0.5, tx=0, ty=0)

var originalAndInverseMatrix:Matrix = doubleScaleMatrix.clone();
originalAndInverseMatrix.concat(halfScaleMatrix);
rectangleTrans_3.matrix = originalAndInverseMatrix;
rectangleMc_3._x = 300;
trace(originalAndInverseMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

Matrix constructor

```
public Matrix([a:Number], [b:Number], [c:Number], [d:Number], [tx:Number], [ty:Number])
```

Creates a new Matrix object with the specified parameters. In matrix notation, the properties will be organized like this:

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

If you do not provide any parameters to the new `Matrix()` constructor, it creates an "identity matrix" with the following values:

a = 1	b = 0
c = 0	d = 1
tx = 0	ty = 0

In matrix notation, the identity matrix looks like this:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Availability
Flash Lite 3.1

Parameters
a : Number [optional] - The value in the first row and first column of the new Matrix object.

b : Number [optional] - The value in the first row and second column of the new Matrix object.

c : Number [optional] - The value in the second row and first column of the new Matrix object.

d : Number [optional] - The value in the second row and second column of the new Matrix object.

tx : Number [optional] - The value in the third row and first column of the new Matrix object.

ty : Number [optional] - The value in the third row and second column of the new Matrix object.

Example
The following example creates `matrix_1` by sending no parameters to the `Matrix` constructor and `matrix_2` by sending parameters to it. The Matrix object `matrix_1`, which is created with no parameters, is an identity Matrix with the values (a=1, b=0, c=0, d=1, tx=0, ty=0).

```
import flash.geom.Matrix;

var matrix_1:Matrix = new Matrix();
trace(matrix_1); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var matrix_2:Matrix = new Matrix(1, 2, 3, 4, 5, 6);
trace(matrix_2); // (a=1, b=2, c=3, d=4, tx=5, ty=6)
```

rotate (Matrix.rotate method)

```
public rotate(angle:Number) : Void
```

Sets the values in the current matrix so that the matrix can be used to apply a rotation transformation.

The `rotate()` method alters the `a` and `d` properties of the Matrix object. In matrix notation this is shown as follows:

$$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Availability
Flash Lite 3.1

Parameters
angle : Number - The rotation angle in radians.

Example
The following example shows how the `rotate()` method rotates `rectangleMc` 30 degrees clockwise. Applying `myMatrix` to `rectangleMc` resets its *_x* value, leaving you to reset it to 100 manually.

```
import flash.geom.Matrix;
import flash.geom.Transform;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var degrees:Number = 30;
var radians:Number = (degrees/180) Math.PI;
myMatrix.rotate(radians);
trace(myMatrix.toString()); // (a=0.866025403784439, b=0.5, c=-0.5, d=0.866025403784439,
tx=0, ty=0)

var rectangleMc:MovieClip = createRectangle(20, 80, 0xFF0000);
trace(rectangleMc._x); // 0
rectangleMc._x = 100;
trace(rectangleMc._x); // 100

var rectangleTrans:Transform = new Transform(rectangleMc);
rectangleTrans.matrix = myMatrix;
trace(rectangleMc._x); // 0
rectangleMc._x = 100;
trace(rectangleMc._x); // 100

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

The previous example uses the _x property of the MovieClip object to position `rectangleMc`. Generally, when dealing with Matrix object positioning, mixing positioning techniques is considered poor format. The previous example written in correct syntax would concatenate a translation Matrix to `myMatrix` to change the horizontal location of `rectangleMc`. The following example demonstrates this.

```
import flash.geom.Matrix;
import flash.geom.Transform;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var degrees:Number = 30;
var radians:Number = (degrees/180) * Math.PI;
myMatrix.rotate(radians);
trace(myMatrix.toString()); // (a=0.866025403784439, b=0.5, c=-0.5, d=0.866025403784439,
tx=0, ty=0)

var translateMatrix:Matrix = new Matrix();
translateMatrix.translate(100, 0);
myMatrix.concat(translateMatrix);
trace(myMatrix.toString()); // (a=0.866025403784439, b=0.5, c=-0.5, d=0.866025403784439,
tx=100, ty=0)

var rectangleMc:MovieClip = createRectangle(20, 80, 0xFF0000);
trace(rectangleMc._x); // 0
rectangleMc._x = 100;
trace(rectangleMc._x); // 100

var rectangleTrans:Transform = new Transform(rectangleMc);
rectangleTrans.matrix = myMatrix;
trace(rectangleMc._x); // 100

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

scale (Matrix.scale method)

```
public scale(sx:Number, sy:Number) : Void
```

Modifies a matrix so that its effect, when applied, is to resize an image. In the resized image, the location of each pixel on the *x* axis is multiplied by `sx`; and on the *y* axis, it is multiplied by `sy`.

The `scale()` method alters the `a` and `d` properties of the matrix object. In matrix notation, this is shown as follows:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Availability
Flash Lite 3.1

Parameters
sx : Number - A multiplier used to scale the object along the *x* axis.

sy : Number - A multiplier used to scale the object along the *y* axis.

Example
The following example uses the `scale()` method to scale `myMatrix` by a factor of three horizontally and a factor of four vertically.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 100, 100);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=100, ty=100)

myMatrix.scale(3, 4);
trace(myMatrix.toString()); // (a=6, b=0, c=0, d=8, tx=300, ty=400)
```

toString (Matrix.toString method)

```
public toString() : String
```

Returns a text value listing the properties of the Matrix object.

Availability
Flash Lite 3.1

Returns
"`String`" on page 584 - A string containing the values of the properties of the Matrix object: `a`, `b`, `c`, `d`, `tx`, and `ty`.

Example
The following example creates `myMatrix` and converts its values to a string in the format of (a=A, b=B, c=C, d=D, tx=TX, ty=TY).

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace("myMatrix: " + myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
```

transformPoint (Matrix.transformPoint method)

```
public transformPoint(pt:Point) : Point
```

Applies the geometric transformation represented by the Matrix object to the specified point.

Availability
Flash Lite 3.1

Parameters
pt : Point - The Point (*x*,*y*) to be transformed.

Returns

Point (flash.geom.Point) - The new Point object.

Example

The following example uses the `transformPoint()` method to create `transformedPoint` from `myPoint`. The `translate()` method does have an effect on the position of `transformedPoint`. In the example, `scale()` increases the original `x` value by a factor of three, from 50 to 150, and the `translate()` method increases `x` by 300, for a total value of 450.

```
import flash.geom.Matrix;
import flash.geom.Point;

var myMatrix:Matrix = new Matrix();
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.translate(100, 0);
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=100, ty=0)

myMatrix.scale(3, 3);
trace(myMatrix); // (a=3, b=0, c=0, d=3, tx=300, ty=0)

var myPoint:Point = new Point(50,0);
trace(myPoint); // (50, 0)

var transformedPoint:Point = myMatrix.transformPoint(myPoint);
trace(transformedPoint); // (450, 0)

var pointMc_0:MovieClip = createRectangle(10, 10, 0xFF0000);
pointMc_0._x = myPoint.x;

var pointMc_1:MovieClip = createRectangle(10, 10, 0x00FF00);
pointMc_1._x = transformedPoint.x;

function createRectangle(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

translate (Matrix.translate method)

```
public translate(tx:Number, ty:Number) : Void
```

Modifies a Matrix object so that the effect of its transformation is to move an object along the *x* and *y* axes.

The `translate()` method alters the `tx` and `ty` properties of the matrix object. In matrix notation, this is shown as follows:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Availability
Flash Lite 3.1

Parameters
tx : Number - The amount of movement along the *x* axis to the right, in pixels.

ty : Number - The amount of movement down along the *y* axis, in pixels.

Example
The following example uses the `translate()` method to position `rectangleMc` x:100 and y:50. The `translate()` method affects the translation properties `tx` and `ty`, but it doesn't affect the `a`, `b`, `c`, or `d` properties.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 100, 100);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=100, ty=100)

myMatrix.translate(100, 50);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=200, ty=150)
```

tx (Matrix.tx property)

```
public tx : Number
```

The distance by which to translate each point along the *x* axis. This represents the value in the third row and first column of the Matrix object.

Availability
Flash Lite 3.1

Example
The following example creates the Matrix object `myMatrix` and sets its `tx` value.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.tx); // 0

myMatrix.tx = 50; // 50
trace(myMatrix.tx);
```

ty (Matrix.ty property)

```
public ty : Number
```

The distance by which to translate each point along the *y* axis. This represents the value in the third row and second column of the Matrix object.

Availability
Flash Lite 3.1

Example
The following example creates the Matrix object `myMatrix` and sets its `ty` value.

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.ty); // 0

myMatrix.ty = 50;
trace(myMatrix.ty); // 50
```

Mouse

```
Object
    |
    +-Mouse

public class Mouse
extends Object
```

The Mouse class is a top-level class whose properties and methods you can access without using a constructor. You can use the methods of the Mouse class to add and remove listeners and to handle mouse events.

The members of this class are supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Property summary
Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Event summary

Event	Description
onMouseDown = function() {}	Notified when the mouse button is pressed.
onMouseMove = function() {}	Notified when the mouse moves.
onMouseUp = function() {}	Notified when the mouse button is released.

Method summary

Modifiers	Signature	Description
static	addListener(listener: Object) : Void	Registers an object to receive notifications of the onMouseDown, onMouseMove, and onMouseUp listeners.
static	removeListener(listener:Object) : Boolean	Removes an object that was previously registered with addListener().

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

addListener (Mouse.addListener method)

```
public static addListener(listener:Object) : Void
```

Registers an object to receive notifications of the onMouseDown, onMouseMove, and onMouseUp listeners.

The listener parameter should contain an object that has a defined method for at least one of the listeners.

When the mouse button is pressed, moved, released, or used to scroll, regardless of the input focus, all listening objects that are registered with this method have their onMouseDown, onMouseMove, or onMouseUp method invoked. Multiple objects can listen for mouse notifications. If the listener is already registered, no change occurs.

Note: This method is supported in Flash Lite only if System.capabilities.hasMouse is true or System.capabilities.hasStylus is true.

Availability

Flash Lite 2.0

Parameters

listener : Object

Example

This example sends the position of the cursor to the Output window.

```
// Create a mouse listener object.
var mouseListener:Object = new Object();

mouseListener.onMouseMove = function() {
    trace(_xmouse);
    trace(_ymouse);
};
Mouse.addListener(mouseListener);
```

See also

onMouseDown (Mouse.onMouseDown event listener), onMouseMove (Mouse.onMouseMove event listener), onMouseUp (Mouse.onMouseUp event listener)

onMouseDown (Mouse.onMouseDown event listener)

```
onMouseDown = function() {}
```

Notified when the mouse button is pressed. To use the `onMouseDown` listener, you must create a listener object. Define a function for `onMouseDown` and call `addListener()` to register the listener with the Mouse object, as shown in the following code:

```
var someListener:Object = new Object();
someListener.onMouseDown = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

This event listener is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example uses the Drawing API to draw a rectangle when the user presses the mouse button, moves the mouse, and then releases the mouse button at runtime.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    this.orig_x = _xmouse;
    this.orig_y = _ymouse;
    this.target_mc = canvas_mc.createEmptyMovieClip("", canvas_mc.getNextHighestDepth());
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        this.target_mc.clear();
        this.target_mc.lineStyle(1, 0xFF0000, 100);
        this.target_mc.moveTo(this.orig_x, this.orig_y);
        this.target_mc.lineTo(_xmouse, this.orig_y);
        this.target_mc.lineTo(_xmouse, _ymouse);
        this.target_mc.lineTo(this.orig_x, _ymouse);
        this.target_mc.lineTo(this.orig_x, this.orig_y);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

See also

addListener (Mouse.addListener method)

onMouseMove (Mouse.onMouseMove event listener)

```
onMouseMove = function() {}
```

Notified when the mouse moves. To use the `onMouseMove` listener, you must create a listener object. You can then define a function for `onMouseMove` and use `addListener()` to register the listener with the Mouse object, as shown in the following code:

```
var someListener:Object = new Object();
someListener.onMouseMove = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

This event listener is supported in Flash Lite only if `System.capabilities.hasMouse` is `true`.

Availability

Flash Lite 2.0

Example

The following example uses the mouse pointer as a tool to draw lines using `onMouseMove` and the Drawing API. The user draws a line by moving the pointer.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

The following example sets the `x` and `y` positions of the `pointer_mc` movie clip instance to the `x` and `y` pointer positions. The device must support a stylus or mouse for this example to work. To use the example, you create a movie clip and set its Linkage identifier to `pointer_id`. Then add the following ActionScript code to Frame 1 of the Timeline:

```
this.attachMovie("pointer_id", "pointer_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    pointer_mc._x = _xmouse;
    pointer_mc._y = _ymouse;
};
Mouse.addListener(mouseListener);
```

See also

addListener (Mouse.addListener method)

onMouseUp (Mouse.onMouseUp event listener)

```
onMouseUp = function() {}
```

Notified when the mouse button is released. To use the `onMouseUp` listener, you must create a listener object. You can then define a function for `onMouseUp` and use `addListener()` to register the listener with the Mouse object, as shown in the following code:

```
var someListener:Object = new Object();
someListener.onMouseUp = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

This event listener is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example uses the mouse pointer as a tool to draw lines using `onMouseMove` and the Drawing API. The user draws a line by moving the pointer and stops drawing the line by releasing the mouse button.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

See also

`addListener (Mouse.addListener method)`

removeListener (Mouse.removeListener method)

```
public static removeListener(listener:Object) : Boolean
```

Removes an object that was previously registered with `addListener()`.

Note: This method is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Parameters

listener : Object

Returns

Boolean - If the listener object is successfully removed, the method returns `true`; if the listener object is not successfully removed (for example, if it was not on the Mouse object's listener list), the method returns `false`.

Example

The following example attaches three buttons to the Stage, and lets the user draw lines in the SWF file at runtime, using the mouse pointer. One button clears all of the lines from the SWF file. The second button removes the mouse listener so the user cannot draw lines. The third button adds the mouse listener after it is removed, so the user can draw lines again. Add the following ActionScript to Frame 1 of the Timeline:

```
this.createClassObject(mx.controls.Button, "clear_button", this.getNextHighestDepth(),
{_x:10, _y:10, label:'clear'});
this.createClassObject(mx.controls.Button, "stopDrawing_button", this.getNextHighestDepth(),
{_x:120, _y:10, label:'stop drawing'});
this.createClassObject(mx.controls.Button, "startDrawing_button",
this.getNextHighestDepth(), {_x:230, _y:10, label:'start drawing'});
startDrawing_button.enabled = false;
//
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
    canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

```
var clearListener:Object = new Object();
clearListener.click = function() {
    canvas_mc.clear();
};
clear_button.addEventListener("click", clearListener);
//
var stopDrawingListener:Object = new Object();
stopDrawingListener.click = function(evt:Object) {
    Mouse.removeListener(mouseListener);
    evt.target.enabled = false;
    startDrawing_button.enabled = true;
};
stopDrawing_button.addEventListener("click", stopDrawingListener);
var startDrawingListener:Object = new Object();
startDrawingListener.click = function(evt:Object) {
    Mouse.addListener(mouseListener);
    evt.target.enabled = false;
    stopDrawing_button.enabled = true;
};
startDrawing_button.addEventListener("click", startDrawingListener);
```

MovieClip

```
Object
    |
    +-MovieClip

public dynamic class MovieClip
extends Object
```

Use the MovieClip class to manipulate movie clips with ActionScript. There is no constructor for the MovieClip class. To create a new movie clip instance, do one of the following:

- Draw a movie clip on the Stage in the Flash authoring tool and give it an instance name in the Property inspector.
- Call `attachMovie()` method to create a new movie clip instance based on a movie clip symbol that exists in the library.
- Call `createEmptyMovieClip()` to create a new, empty movie clip instance as a child based on another movie clip.
- Call `duplicateMovieClip()` method to create a movie clip instance based on another movie clip.

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
	_alpha : Number	The alpha transparency value of the movie clip.
	_currentframe : Number [read-only]	Returns the number of the frame in which the playhead is located in the movie clip's Timeline.
	_droptarget : String [read-only]	Returns the absolute path in slash-syntax notation of the movie clip instance on which this movie clip was dropped.

Modifiers	Property	Description
	enabled : Boolean	A Boolean value that indicates whether a movie clip is enabled.
	focusEnabled : Boolean	If the value is `undefined` or `false`, a movie clip cannot receive input focus unless it is a button.
	_focusrect : Boolean	A Boolean value that specifies whether a movie clip has a yellow rectangle around it when it has input focus.
	_framesloaded : Number [read-only]	The number of frames that are loaded from a streaming SWF file.
	_height : Number	The height of the movie clip, in pixels.
	_highquality : Number	**Deprecated** since Flash Player 7. This property was deprecated in favor of `MovieClip._quality`. Specifies the level of anti-aliasing applied to the current SWF file.
	hitArea : Object	Designates another movie clip to serve as the hit area for a movie clip.
	_lockroot : Boolean	A Boolean value that specifies what `_root` refers to when a SWF file is loaded into a movie clip.
	_name : String	The instance name of the movie clip.
	_parent : MovieClip	A reference to the movie clip or object that contains the current movie clip or object.
	_quality : String	Sets or retrieves the rendering quality used for a SWF file.
	_rotation : Number	Specifies the rotation of the movie clip, in degrees, from its original orientation.
	_soundbuftime : Number	Specifies the number of seconds a sound prebuffers before it starts to stream.
	tabChildren : Boolean	Determines whether the children of a movie clip are included in the automatic tab ordering.
	tabEnabled : Boolean	Specifies whether the movie clip is included in automatic tab ordering.
	tabIndex : Number	Lets you customize the tab ordering of objects in a movie.
	_target : String [read-only]	Returns the target path of the movie clip instance, in slash notation.
	_totalframes : Number [read-only]	Returns the total number of frames in the movie clip instance specified in the `MovieClip` parameter.
	trackAsMenu : Boolean	A Boolean value that indicates whether other buttons or movie clips can receive a release event from a mouse or stylus.
	_url : String [read-only]	Retrieves the URL of the SWF, JPEG, GIF, or PNG file from which the movie clip was downloaded.
	_visible : Boolean	A Boolean value that indicates whether the movie clip is visible.
	_width : Number	The width of the movie clip, in pixels.
	_x : Number	An integer that sets the *x* coordinate of a movie clip relative to the local coordinates of the parent movie clip.

Modifiers	Property	Description
	_xmouse : Number [read-only]	Returns the *x* coordinate of the mouse position.
	_xscale : Number	Sets the horizontal scale (`percentage`) of the movie clip as applied from the registration point of the movie clip.
	_y : Number	Sets the *y* coordinate of a movie clip relative to the local coordinates of the parent movie clip.
	_ymouse : Number [read-only]	Indicates the *y* coordinate of the mouse position.
	_yscale : Number	Sets the vertical scale (`percentage`) of the movie clip as applied from the registration point of the movie clip.

Properties inherited from class Object

```
constructor (Object.constructor property), __proto__ (Object.__proto__ property),
prototype (Object.prototype property), __resolve (Object.__resolve property)
```

Event summary

Event	Description
onData = function() {}	Invoked when a movie clip receives data from a `MovieClip.loadVariables()` or `MovieClip.loadMovie()` call.
onDragOut = function() {}	Invoked when the mouse button is pressed and the pointer rolls outside the object.
onDragOver = function() {}	Invoked when the pointer is dragged outside and then over the movie clip.
onEnterFrame = function() {}	Invoked repeatedly at the frame rate of the SWF file.
onKeyDown = function() {}	Invoked when a movie clip has input focus and a key is pressed.
onKeyUp = function() {}	Invoked when a key is released.
onKillFocus = function(newFocus: Object) {}	Invoked when a movie clip loses input focus.
onLoad = function() {}	Invoked when the movie clip is instantiated and appears in the Timeline.
onMouseDown = function() {}	Invoked when the mouse button is pressed.
onMouseMove = function() {}	Invoked when the mouse moves.
onMouseUp = function() {}	Invoked when the mouse button is released.
onPress = function() {}	Invoked when the user clicks the mouse while the pointer is over a movie clip.
onRelease = function() {}	Invoked when the mouse button is released over a movie clip.

Event	Description
onReleaseOutside = function() {}	Invoked when the mouse button is pressed inside the movie clip area and then released outside the movie clip area.
onRollOut = function() {}	Invoked when the pointer moves outside a movie clip area.
onRollOver = function() {}	Invoked when the pointer moves over a movie clip area.
onSetFocus = function(oldFocus: Object) {}	Invoked when a movie clip receives input focus.
onUnload = function() {}	Invoked in the first frame after the movie clip is removed from the Timeline.

Method summary

Modifiers	Signature	Description
	attachMovie(id:String, name:String, depth:Number, [initObject:Object]) : MovieClip	Takes a symbol from the library and attaches it to the movie clip.
	beginFill(rgb:Number, [alpha:Number]) : Void	Indicates the beginning of a new drawing path.
	beginGradientFill(fillType:String, colors:Array, alphas:Array, ratios:Array, matrix:Object) : Void	Indicates the beginning of a new drawing path.
	clear() : Void	Removes all the graphics created during runtime by using the movie clip draw methods, including line styles specified with MovieClip.lineStyle().
	createEmptyMovieClip(name:String, depth:Number) : MovieClip	Creates an empty movie clip as a child of an existing movie clip.
	createTextField(instanceName:String, depth:Number, x:Number, y:Number, width:Number, height:Number) : TextField	Creates a new, empty text field as a child of the movie clip on which you call this method.
	curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number) : Void	Draws a curve using the current line style from the current drawing position to (anchorX, anchorY) using the control point that (controlX, controlY) specifies.

Modifiers	Signature	Description
	duplicateMovieClip(name:String, depth:Number, [initObject:Object]) : MovieClip	Creates an instance of the specified movie clip while the SWF file is playing.
	endFill() : Void	Applies a fill to the lines and curves added since the last call to beginFill() or beginGradientFill().
	getBounds(bounds:Object) : Object	Returns properties that are the minimum and maximum *x* and *y* coordinate values of the movie clip, based on the bounds parameter.
	getBytesLoaded() : Number	Returns the number of bytes that have already loaded (streamed) for the movie clip.
	getBytesTotal() : Number	Returns the size, in bytes, of the movie clip.
	getDepth() : Number	Returns the depth of the movie clip instance.
	getInstanceAtDepth(depth:Number) : MovieClip	Determines if a particular depth is already occupied by a movie clip.
	getNextHighestDepth() : Number	Determines a depth value that you can pass to MovieClip.attachMovie(), MovieClip.duplicateMovieClip(), or MovieClip.createEmptyMovieClip() to ensure that Flash Lite renders the movie clip in front of all other objects on the same level and layer in the current movie clip.
	getSWFVersion() : Number	Returns an integer that indicates the Flash Lite publish version for the movie clip.
	getURL(url:String, [window:String], [method:String]) : Void	Loads a document from the specified URL into the specified window.
	globalToLocal(pt:Object) : Void	Converts the pt object from Stage (global) coordinates to the movie clip's (local) coordinates.
	gotoAndPlay(frame:Object) : Void	Starts playing the SWF file at the specified frame.
	gotoAndStop(frame:Object) : Void	Brings the playhead to the specified frame of the movie clip and stops it there.
	hitTest() : Boolean	Evaluates the movie clip to see if it overlaps or intersects with the hit area that the target or x and y coordinate parameters identify.
	lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean, noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number) : Void	Specifies a line style that for subsequent calls to lineTo() and curveTo() until you call lineStyle() with different parameters.

Modifiers	Signature	Description
	lineTo(x:Number, y:Number) : Void	Draws a line using the current line style from the current drawing position to (*x*, *y*); the current drawing position is then set to (*x*, *y*).
	loadMovie(url:String, [method:String]) : Void	Loads SWF or JPEG files into a movie clip in Flash Lite while the original SWF file is playing.
	loadVariables(url:String , [method:String]) : Void	Reads data from an external file and sets the values for variables in the movie clip.
	localToGlobal(pt:Object) : Void	Converts the *pt* object from the movie clip's (local) coordinates to the Stage (global) coordinates.
	moveTo(x:Number, y:Number) : Void	Moves the current drawing position to (*x*, *y*).
	nextFrame() : Void	Sends the playhead to the next frame and stops it.
	play() : Void	Moves the playhead in the Timeline of the movie clip.
	prevFrame() : Void	Sends the playhead to the previous frame and stops it.
	removeMovieClip() : Void	Removes a movie clip instance created with `duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`, `MovieClip.createEmptyMovieClip()`, or `MovieClip.attachMovie()`.
	setMask(mc:Object) : Void	Makes the movie clip in the parameter *mc* a mask that reveals the calling movie clip.
	startDrag([lockCenter:Boolean], [left:Number], [top:Number], [right:Number], [bottom:Number]) : Void	Lets the user drag the specified movie clip.
	stop() : Void	Stops the movie clip currently playing.
	stopDrag() : Void	Ends a call to the MovieClip.startDrag() method.
	swapDepths(target:Object) : Void	Swaps the stacking, or depth level (z-order), of this movie clip with the movie clip specified by the `target` parameter, or with the movie clip that currently occupies the depth level specified in the `target` parameter.
	unloadMovie() : Void	Removes the contents of a movie clip instance.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

_alpha (MovieClip._alpha property)

```
public _alpha : Number
```

The alpha transparency value of a movie clip. Valid values are 0 (fully transparent) to 100 (fully opaque). The default value is 100. Objects in a movie clip with `_alpha` set to 0 are active, even though they are invisible. For example, you can still click a button in a movie clip whose `_alpha` property is set to 0. To disable the button completely, you can set the movie clip's `_visible` property to false.

Availability

Flash Lite 2.0

Example

The following code sets the `_alpha` property of a movie clip named `rect_mc` to 50% when you press a button called my_btn.

```
my_btn.onPress = function(){
     rect_mc._alpha  =   50 ;
}
my_btn.onRelease = function(){
    rect_mc._alpha = 100;
}
```

See also

`_alpha (Button._alpha property)`, `_alpha (TextField._alpha property)`, `_visible (MovieClip._visible property)`

attachBitmap (MovieClip.attachBitmap method)

```
public attachBitmap(bmp:BitmapData, depth:Number, [pixelSnapping:String],
[smoothing:Boolean]) : Void
```

Attaches a bitmap image to a movie clip.

After the bitmap is attached to the movie clip, a reference is made from the movie clip to the bitmap object. When attaching a bitmap, you can specify `pixelSnapping` and `smoothing` parameters to affect the appearance of the bitmap.

After a bitmap is added to the movie clip, it is not an accessible object. The `depth`, `pixelSnapping`, and `smoothing` parameters can only be set during the `attachBitmap()` method call and cannot be changed later.

First use the `createEmptyMovieClip()` to create an empty movie clip, then call the `attachBitmap()` method. This way, you can apply transformations to the movie clip to transform the bitmap; for example, you can call the `matrix` property of the movie clip.

Pixel snapping forces the position of the bitmap to the nearest whole pixel value instead of positioning to be on a partial pixel. There are three pixel snapping modes:

- Auto mode does pixel snapping as long as the bitmap is not stretched or rotated.
- Always mode always does pixel snapping, regardless of stretching and rotation.
- Never mode turns off pixel snapping for the movie clip.

Smoothing mode affects the appearance of the image when it is scaled.

Parameters

bmp`:flash.display.BitmapData` - A transparent or opaque bitmap image.

depth : `Number` - An integer that specifies the depth level within the movie clip where the bitmap image should be placed.

pixelSnapping : `String` [optional] - The pixel snapping modes are `auto`, `always`, and `never`. The default mode is `auto`.

smoothing : `Boolean` [optional] - The smoothing mode is either `true` for enabled or `false` for disabled. The default mode is disabled.

Availability

Flash Lite 3.1

Example

The following code creates a BitmapData object and attaches it to a movie clip:

```
import flash.display.*;
this.createEmptyMovieClip("bmp1", 99);
var bmpData1:BitmapData = new BitmapData(200, 200, false, 0xaa3344);
bmp1.attachBitmap(bmpData1, 2, "auto", true);
```

attachMovie (MovieClip.attachMovie method)

```
public attachMovie(id:String, name:String, depth:Number, [initObject:Object]) : MovieClip
```

Takes a symbol from the library and attaches it to the movie clip. Use `MovieClip.removeMovieClip()` or `MovieClip.unloadMovie()` to remove a symbol attached with `attachMovie()`.

Availability

Flash Lite 2.0

Parameters

id : String - The linkage name of the movie clip symbol in the library to attach to a movie clip on the Stage. This is the name entered in the Identifier field in the Linkage Properties dialog box.

name : String - A unique instance name for the movie clip being attached to the movie clip.

depth : Number - An integer specifying the depth level where the SWF file is placed.

initObject : Object [optional] - (Supported for Flash Player 6 and later) An object containing properties with which to populate the newly attached movie clip. This parameter allows dynamically created movie clips to receive clip parameters. If `initObject` is not an object, it is ignored. All properties of `initObject` are copied into the new instance. The properties specified with `initObject` are available to the constructor function.

Returns

MovieClip - A reference to the newly created instance.

Example

The following example attaches two instances of a symbol with the linkage identifier "circle" to a movie clip instance on the Stage:

```
this.attachMovie("circle", "circle1_mc", this.getNextHighestDepth());
this.attachMovie("circle", "circle2_mc", this.getNextHighestDepth(), {_x:50, _y:50});
```

See also

removeMovieClip (MovieClip.removeMovieClip method), unloadMovie (MovieClip.unloadMovie method), removeMovieClip function

beginFill (MovieClip.beginFill method)

```
public beginFill(rgb:Number, [alpha:Number]) : Void
```

Indicates the beginning of a new drawing path. If an open path exists (that is, if the current drawing position does not equal the previous position specified in a `MovieClip.moveTo()` method) and a fill is associated with it, that path is closed with a line and then filled. This is similar to what happens when `MovieClip.endFill()` is called.

Availability

Flash Lite 2.0

Parameters

rgb : Number - A hex color value (for example, red is 0xFF0000, blue is 0x0000FF, and so on). If this value is not provided or is undefined, a fill is not created.

alpha : Number [optional] - An integer from 0 to 100 that specifies the alpha value of the fill. If this value is not provided, 100 (solid) is used. If the value is less than 0, Flash uses 0. If the value is greater than 100, Flash uses 100.

Example

The following example creates a square with red fill on the Stage:

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());
square_mc.beginFill(0xFF0000);
square_mc.moveTo(10, 10);
square_mc.lineTo(100, 10);
square_mc.lineTo(100, 100);
square_mc.lineTo(10, 100);
square_mc.lineTo(10, 10);
square_mc.endFill();
```

See also

moveTo (MovieClip.moveTo method), endFill (MovieClip.endFill method), beginGradientFill (MovieClip.beginGradientFill method)

beginGradientFill (MovieClip.beginGradientFill method)

```
public beginGradientFill(fillType:String, colors:Array, alphas:Array, ratios:Array,
matrix:Object) : Void
```

Indicates the beginning of a new drawing path. If the first parameter is `undefined`, or if no parameters are passed, the path has no fill. If an open path exists (that is, if the current drawing position does not equal the previous position specified in a `MovieClip.moveTo()` method), and it has a fill associated with it, that path is closed with a line and then filled. This is similar to what happens when you call `MovieClip.endFill()`.

This method fails if any of the following conditions exist:

- The number of items in the `colors`, `alphas`, and `ratios` parameters are not equal.
- The `fillType` parameter is not `"linear"` or `"radial"`.
- Any of the fields in the object for the `matrix` parameter are missing or invalid.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Parameters
fillType : String - Either the string "`linear`" or the string "`radial`".

colors : Array - An array of RGB hex color values to be used in the gradient (for example, red is 0xFF0000, blue is 0x0000FF, and so on).

alphas : Array - An array of alpha values for the corresponding colors in the `colors` array; valid values are 0-100. If the value is less than 0, Flash uses 0. If the value is greater than 100, Flash uses 100.

ratios : Array - An array of color distribution ratios; valid values are 0-255. This value defines the percentage of the width where the color is sampled at 100 percent.

matrix : Object - A transformation matrix that is an object with either of the following two sets of properties:

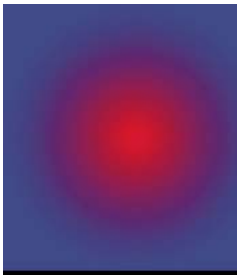
- `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, which can be used to describe a 3 x 3 matrix of the following form:

```
a b c
d e f
g h i
```

  The following example uses the `beginGradientFill()` method with a `matrix` parameter of this type:

```
this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());

gradient_mc._x = -100;
gradient_mc._y = -100;

with (gradient_mc)
{
    colors = [0xFF0000, 0x0000FF];
    fillType = "radial"
    alphas = [100, 100];
    ratios = [0, 0xFF];
    matrix = {a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1};
    beginGradientFill(fillType, colors, alphas, ratios, matrix);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
}
```

This code draws the following image on the screen:



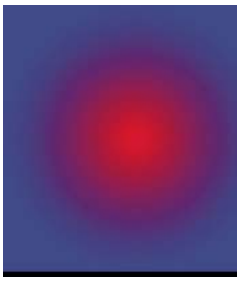
- `matrixType`, `x`, `y`, `w`, `h`, `r`.

  The properties indicate the following: `matrixType` is the string `"box"`, `x` is the horizontal position relative to the registration point of the parent clip for the upper-left corner of the gradient, `y` is the vertical position relative to the registration point of the parent clip for the upper-left corner of the gradient, `w` is the width of the gradient, `h` is the height of the gradient, and `r` is the rotation in radians of the gradient.

  The following example uses the `beginGradientFill()` method with a `matrix` parameter of this type:

```
this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());

gradient_mc._x = -100;
gradient_mc._y = -100;

with (gradient_mc)
{
    colors = [0xFF0000, 0x0000FF];
    fillType = "radial"
    alphas = [100, 100];
    ratios = [0, 0xFF];
    matrix = {matrixType:"box", x:100, y:100, w:200, h:200,
        r:(45/180)*Math.PI};
    beginGradientFill(fillType, colors, alphas, ratios, matrix);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
}
```

This code draws the following image on the screen:



See also

beginFill (MovieClip.beginFill method), endFill (MovieClip.endFill method), lineStyle (MovieClip.lineStyle method), lineTo (MovieClip.lineTo method), moveTo (MovieClip.moveTo method)

clear (MovieClip.clear method)

```
public clear() : Void
```

Removes all the graphics created during runtime by using the movie clip draw methods, including line styles specified with `MovieClip.lineStyle()`. Shapes and lines that are manually drawn during authoring time (with the Flash drawing tools) are unaffected.

Availability

Flash Lite 2.0

Example

The following example draws a box on the Stage. When the user clicks a button called removeBox_btn, the graphic is removed.

```
this.createEmptyMovieClip("box_mc", 1);
drawBox(box_mc, 10, 10, 100, 100);
function drawBox(mc:MovieClip, x:Number, y:Number, w:Number, h:Number):Void {
    mc.lineStyle(5);
    mc.beginFill(0x009999);
    mc.moveTo(x, y);
    mc.lineTo(x+w, y);
    mc.lineTo(x+w, y+h);
    mc.lineTo(x, y+h);
    mc.lineTo(x, y);
    mc.endFill();
}
removeBox_btn.onRelease = function(){
    box_mc.clear();
}
```

See also

lineStyle (MovieClip.lineStyle method)

createEmptyMovieClip (MovieClip.createEmptyMovieClip method)

```
public createEmptyMovieClip(name:String, depth:Number) : MovieClip
```

Creates an empty movie clip as a child of an existing movie clip. This method behaves similarly to the `attachMovie()` method, but you don't need to provide an external linkage identifier for the new movie clip. The registration point for a newly created empty movie clip is the upper-left corner. This method fails if any of the parameters are missing.

Availability

Flash Lite 2.0

Parameters

name : String - A string that identifies the instance name of the new movie clip.

depth : Number - An integer that specifies the depth of the new movie clip.

Returns

MovieClip - A reference to the newly created movie clip.

Example

The following example creates an empty MovieClip named `container`, creates a new TextField inside of it, and then sets the new `TextField.text` property.

```
var container:MovieClip = this.createEmptyMovieClip("container", this.getNextHighestDepth());
var label:TextField = container.createTextField("label", 1, 0, 0, 150, 20);
label.text = "Hello World";
```

See also

attachMovie (MovieClip.attachMovie method)

createTextField (MovieClip.createTextField method)

```
public createTextField(instanceName:String, depth:Number, x:Number, y:Number, width:Number,
height:Number) : TextField
```

Creates a new, empty text field as a child of the movie clip on which you call this method. You can use the `createTextField()` method to create text fields while a SWF file plays. The `depth` parameter determines the new text field's depth level (z-order position) in the movie clip. Each depth level can contain only one object. If you create a new text field on a depth that already has a text field, the new text field replaces the existing text field. To avoid overwriting existing text fields, use `MovieClip.getInstanceAtDepth()` to determine whether a specific depth is already occupied, or `MovieClip.getNextHighestDepth()`, to determine the highest unoccupied depth. The text field is positioned at (*x*, *y*) with dimensions *width* by *height*. The `x` and `y` parameters are relative to the container movie clip; these parameters correspond to the `_x` and `_y` properties of the text field. The `width` and `height` parameters correspond to the `_width` and `_height` properties of the text field.

The default properties of a text field are as follows:

```
type = "dynamic"
border = false
background = false
password = false
multiline = false
html = false
embedFonts = false
selectable = true
wordWrap = false
mouseWheelEnabled = true
condenseWhite = false
restrict = null
variable = null
maxChars = null
styleSheet = undefined
tabInded = undefined
```

A text field created with `createTextField()` receives the following default TextFormat object settings:

```
font = "Times New Roman" // "Times" on Mac OS
size = 12
color = 0x000000
bold = false
italic = false
underline = false
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
blockIndent = 0
bullet = false
display = block
tabStops = [] // (empty array)
```

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

instanceName : String - A string that identifies the instance name of the new text field.

depth : Number - A positive integer that specifies the depth of the new text field.

x : Number - An integer that specifies the *x* coordinate of the new text field.

y : Number - An integer that specifies the *y*coordinate of the new text field.

width : Number - A positive integer that specifies the width of the new text field.

height : Number - A positive integer that specifies the height of the new text field.

Returns

TextField

Example

The following example creates a text field with a width of 300, a height of 100, an *x* coordinate of 100, a *y* coordinate of 100, no border, red, and underlined text:

```
this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
var my_fmt:TextFormat = new TextFormat();
my_fmt.color = 0xFF0000;
my_fmt.underline = true;
my_txt.text = "This is my first test field object text.";
my_txt.setTextFormat(my_fmt);
```

An example is also in the animations.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

getInstanceAtDepth (MovieClip.getInstanceAtDepth method), getNextHighestDepth (MovieClip.getNextHighestDepth method), getNewTextFormat (TextField.getNewTextFormat method)

_currentframe (MovieClip._currentframe property)

```
public _currentframe : Number [read-only]
```

Returns the number of the frame in which the playhead is located in the movie clip's Timeline.

Availability

Flash Lite 2.0

Example

The following example uses the `_currentframe` property to direct the playhead of the `actionClip_mc` movie clip to advance five frames ahead of its current location:

```
actionClip_mc.gotoAndStop(actionClip_mc._currentframe + 5);
```

curveTo (MovieClip.curveTo method)

```
public curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number) : Void
```

Draws a curve using the current line style from the current drawing position to (`anchorX`, `anchorY`) using the control point that (`controlX`, `controlY`) specifies. The current drawing position is then set to (`anchorX`, `anchorY`). If the movie clip you are drawing in contains content created with the Flash drawing tools, calls to `curveTo()` are drawn underneath this content. If you call the `curveTo()` method before any calls to the `moveTo()` method, the current drawing position defaults to (0,0). If any of the parameters are missing, this method fails and the current drawing position is not changed.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

controlX : Number - An integer that specifies the horizontal position of the control point relative to the registration point of the parent movie clip.

controlY : Number - An integer that specifies the vertical position of the control point relative to the registration point of the parent movie clip.

anchorX : Number - An integer that specifies the horizontal position of the next anchor point relative to the registration point of the parent movie clip.

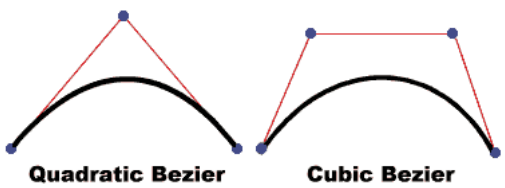
anchorY : Number - An integer that specifies the vertical position of the next anchor point relative to the registration point of the parent movie clip.

Example

The following example draws a nearly circular curve with a solid blue hairline stroke and a solid red fill:

```
this.createEmptyMovieClip("circle_mc", 1);
with (circle_mc) {
    lineStyle(0, 0x0000FF, 100);
    beginFill(0xFF0000);
    moveTo(0, 100);
    curveTo(0,200,100,200);
    curveTo(200,200,200,100);
    curveTo(200,0,100,0);
    curveTo(0,0,0,100);
    endFill();
}
```

The curve drawn in this example is a quadratic Bezier curve. Quadratic Bezier curves consist of two anchor points and a control point. The curve interpolates the two anchor points, and curves toward the control point.



The following script uses the `curveTo()` method and the Math class to create a circle:

```
this.createEmptyMovieClip("circle2_mc", 2);
circle2_mc.lineStyle(0, 0x000000);
drawCircle(circle2_mc, 100, 100, 100);
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, '+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x,
-Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x,
-Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

An example is also in the drawingapi.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

beginFill (MovieClip.beginFill method), createEmptyMovieClip (MovieClip.createEmptyMovieClip method), endFill (MovieClip.endFill method), lineStyle (MovieClip.lineStyle method), lineTo (MovieClip.lineTo method), moveTo (MovieClip.moveTo method), Math

_droptarget (MovieClip._droptarget property)

```
public _droptarget : String [read-only]
```

Returns the absolute path in slash-syntax notation of the movie clip instance on which this movie clip was dropped. The `_droptarget` property always returns a path that starts with a slash (`/`). To compare the `_droptarget` property of an instance to a reference, use the `eval()` function to convert the returned value from slash syntax to a dot-syntax reference (ActionScript 2.0 does not support slash syntax.)

Note: This property is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example evaluates the `_droptarget` property of the `garbage_mc` movie clip instance and uses `eval()` to convert it from slash syntax to a dot syntax reference. The `garbage_mc` reference is then compared to the reference to the `trashcan_mc` movie clip instance. If the two references are equivalent, the visibility of `garbage_mc` is set to `false`. If they are not equivalent, the `garbage` instance resets to its original position.

```
origX = garbage_mc._x;
origY = garbage_mc._y;
garbage_mc.onPress = function() {
    this.startDrag();
};
garbage_mc.onRelease = function() {
    this.stopDrag();
    if (eval(this._droptarget) == trashcan_mc) {
    this._visible = false;
    } else {
    this._x = origX;
    this._y = origY;
    }
};
```

See also

startDrag (MovieClip.startDrag method), stopDrag (MovieClip.stopDrag method), eval function

duplicateMovieClip (MovieClip.duplicateMovieClip method)

```
public duplicateMovieClip(name:String, depth:Number, [initObject:Object]) : MovieClip
```

Creates an instance of the specified movie clip while the SWF file is playing. Duplicated movie clips always start playing at Frame 1, no matter what frame the original movie clip is on when the `duplicateMovieClip()` method is called. Variables in the parent movie clip are not copied into the duplicate movie clip. Movie clips that are created with the `duplicateMovieClip()` method are not duplicated if you call the `duplicateMovieClip()` method on their parent. If the parent movie clip is deleted, the duplicate movie clip is also deleted. If you used `MovieClip.loadMovie()` or the MovieClipLoader class to load a movie clip, the contents of the SWF file are not duplicated. This means that you cannot save bandwidth by loading a JPEG, GIF, PNG, or SWF file and then duplicating the movie clip.

Contrast this method with the global function version of `duplicateMovieClip()`. The global version of this method requires a parameter that specifies the target movie clip to duplicate. Such a parameter is unnecessary for the MovieClip class version, because the target of the method is the movie clip instance on which the method is invoked. Moreover, the global version of `duplicateMovieClip()` supports neither the `initobject` parameter nor the return value of a reference to the newly created MovieClip instance.

Availability

Flash Lite 2.0

Parameters

name : String - A unique identifier for the duplicate movie clip.

depth : Number - A unique integer specifying the depth at which the new movie clip is placed. Use depth -16384 to place the new movie clip instance beneath all content created in the authoring environment. Values between -16383 and -1, inclusive, are reserved for use by the authoring environment and should not be used with this method. The remaining valid depth values range from 0 to 1048575, inclusive.

initObject : Object [optional] - (Supported for Flash Player 6 and later.) An object containing properties with which to populate the duplicated movie clip. This parameter allows dynamically created movie clips to receive clip parameters. If `initObject` is not an object, it is ignored. All properties of `initObject` are copied into the new instance. The properties specified with `initObject` are available to the constructor function.

Returns

MovieClip - A reference to the duplicated movie clip (supported for Flash Player 6 and later).

Example

The following example duplicates a newly created MovieClip a number of times and traces the target for each duplicate.

```
var container:MovieClip = setUpContainer();
var ln:Number = 10;
var spacer:Number = 1;
var duplicate:MovieClip;
for(var i:Number = 1; i < ln; i++) {
    var newY:Number = i * (container._height + spacer);
    duplicate = container.duplicateMovieClip("clip-" + i, i, {_y:newY});
    trace(duplicate); // _level0.clip-[number]
}

function setUpContainer():MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip("container", this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 20;
    mc.beginFill(0x333333);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    return mc;
}
```

See also

loadMovie (MovieClip.loadMovie method), removeMovieClip (MovieClip.removeMovieClip method), duplicateMovieClip function

enabled (MovieClip.enabled property)

```
public enabled : Boolean
```

A Boolean value that indicates whether a movie clip is enabled. The default value of `enabled` is `true`. If `enabled` is set to `false`, the movie clip's callback methods and `on`*`action`* event handlers are no longer invoked, and the Over, Down, and Up frames are disabled. The `enabled` property does not affect the Timeline of the movie clip; if a movie clip is playing, it continues to play. The movie clip continues to receive movie clip events (for example, `mouseDown`, `mouseUp`, `keyDown`, and `keyUp`).

The `enabled` property only governs the button-like properties of a movie clip. You can change the `enabled` property at any time; the modified movie clip is immediately enabled or disabled. The `enabled` property can be read out of a prototype object. If `enabled` is set to `false`, the object is not included in automatic tab ordering.

Availability

Flash Lite 2.0

Example

The following example disables the `circle_mc` movie clip when the user clicks it:

```
circle_mc.onRelease = function() {
    trace("disabling the "+this._name+" movie clip.");
    this.enabled = false;
};
```

endFill (MovieClip.endFill method)

```
public endFill() : Void
```

Applies a fill to the lines and curves added since the last call to `beginFill()` or `beginGradientFill()`. Flash uses the fill that was specified in the previous call to `beginFill()` or `beginGradientFill()`. If the current drawing position does not equal the previous position specified in a `moveTo()` method and a fill is defined, the path is closed with a line and then filled.

Availability

Flash Lite 2.0

Example

The following example creates a square with red fill on the Stage:

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());
square_mc.beginFill(0xFF0000);
square_mc.moveTo(10, 10);
square_mc.lineTo(100, 10);
square_mc.lineTo(100, 100);
square_mc.lineTo(10, 100);
square_mc.lineTo(10, 10);
square_mc.endFill();
```

An example is also in the drawingapi.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

`beginFill (MovieClip.beginFill method)`, `beginGradientFill (MovieClip.beginGradientFill method)`, `moveTo (MovieClip.moveTo method)`

focusEnabled (MovieClip.focusEnabled property)

```
public focusEnabled : Boolean
```

If the value is `undefined` or `false`, a movie clip cannot receive input focus unless it is a button. If the `focusEnabled` property value is `true`, a movie clip can receive input focus even if it is not a button.

Availability

Flash Lite 2.0

Example

The following example sets the `focusEnabled` property for the movie clip `my_mc` to `false`:

```
my_mc.focusEnabled = false;
```

_focusrect (MovieClip._focusrect property)

```
public _focusrect : Boolean
```

A Boolean value that specifies whether a movie clip has a yellow rectangle around it when it has input focus. This property can override the global `_focusrect` property. The default value of the `_focusrect` property of a movie clip instance is `null`; the movie clip instance does not override the global `_focusrect` property. If the `_focusrect` property of a movie clip instance is set to `true` or `false`, it overrides the setting of the global `_focusrect` property for the single movie clip instance.

Note: For Flash Lite 2.0, when the `_focusrect` property is disabled (in other words, `MovieClip._focusrect` is set to `false`), the movie clip still receives all key press and mouse events.

Also for Flash Lite 2.0, you can change the color of the focus rectangle using the fscommand2 `SetFocusRectColor` command. This behavior is different from Flash Lite player, for which the color of the focus rectangle is restricted to yellow.

Availability

Flash Lite 2.0

Example

This example demonstrates how to hide the yellow rectangle around a specified movie clip instance in a SWF file when the instance has focus in a browser window. Create three movie clips called `mc1_mc`, `mc2_mc`, and `mc3_mc`, and add the following ActionScript to Frame 1 of the Timeline:

```
mc1_mc._focusrect = true;
mc2_mc._focusrect = false;
mc3_mc._focusrect = true;

mc1_mc.onRelease = traceOnRelease;
mc3_mc.onRelease = traceOnRelease;

function traceOnRelease() {
    trace(this._name);
}
```

To test the SWF file in a browser window, select File > Publish Preview > HTML. To give the SWF focus, click it in the browser window and press Tab to focus each instance. You cannot execute code for this movie clip in the browser by pressing Enter or the Spacebar when `_focusrect` is disabled.

You can also test your SWF file in the test environment. Select Control > Disable Keyboard Shortcuts in the test environment. This allows you to view the focus rectangle around the instances in the SWF file.

See also

`_focusrect property`, `_focusrect (Button._focusrect property)`

_framesloaded (MovieClip._framesloaded property)

```
public _framesloaded : Number [read-only]
```

The number of frames that are loaded from a streaming SWF file. This property is useful for determining whether the contents of a specific frame, and all the frames before it, are loaded and are available locally in the browser. It is also useful for monitoring the downloading of large SWF files. For example, you might want to display a message to users indicating that the SWF file is loading until a specified frame in the SWF file has finished loading.

Availability
Flash Lite 2.0

Example
The following example uses the `_framesloaded` property to start a SWF file when all the frames are loaded. If all the frames aren't loaded, the `_xscale` property of the `bar_mc` movie clip instance is increased proportionally to create a progress bar.

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded < this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also
`MovieClipLoader`

getBounds (MovieClip.getBounds method)

```
public getBounds(bounds:Object) : Object
```

Returns properties that are the minimum and maximum *x* and *y* coordinate values of the movie clip, based on the *bounds* parameter.

Note: Use `MovieClip.lcalToGlobal()` and `MovieClip.globalToLocal()` to convert the movie clip's local coordinates to Stage coordinates, or Stage coordinates to local coordinates, respectively.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Parameters
bounds : Object - The target path of the Timeline whose coordinate system you want to use as a reference point.

Returns
Object - An object with the properties `xMin`, `xMax`, `yMin`, and `yMax`.

Example
The following example creates a movie clip called `square_mc`. The code draws a square for that movie clip and uses `MovieClip.getBounds()` to display the coordinate values of the instance in the Output panel.

```
this.createEmptyMovieClip("square_mc", 1);
square_mc._x = 10;
square_mc._y = 10;
square_mc.beginFill(0xFF0000);
square_mc.moveTo(0, 0);
square_mc.lineTo(100, 0);
square_mc.lineTo(100, 100);
square_mc.lineTo(0, 100);
square_mc.lineTo(0, 0);
square_mc.endFill();

var bounds_obj:Object = square_mc.getBounds(this);
for (var i in bounds_obj) {
    trace(i+" --> "+bounds_obj[i]);
}
```

The following information appears in the Output panel:

```
yMax --> 110
yMin --> 10
xMax --> 110
xMin --> 10
```

See also

`globalToLocal (MovieClip.globalToLocal method)`, `localToGlobal (MovieClip.localToGlobal method)`

getBytesLoaded (MovieClip.getBytesLoaded method)

```
public getBytesLoaded() : Number
```

Returns the number of bytes that have already loaded (streamed) for the movie clip. You can compare this value with the value returned by `MovieClip.getBytesTotal()` to determine what percentage of a movie clip has loaded.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Returns

Number - An integer indicating the number of bytes loaded.

Example

The following example uses the `_framesloaded` property to start a SWF file when all the frames are loaded. If all the frames aren't loaded, the `_xscale` property of the `loader` movie clip instance is increased proportionally to create a progress bar.

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/this.getBytesTotal() * 100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

Place your content on or after Frame 3, and then add the following code to Frame 3:

```
stop();
```

See also

getBytesTotal (MovieClip.getBytesTotal method)

getBytesTotal (MovieClip.getBytesTotal method)

```
public getBytesTotal() : Number
```

Returns the size, in bytes, of the movie clip. For movie clips that are external (the root SWF file or a movie clip that is being loaded into a target or a level), the return value is the uncompressed size of the SWF file.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Returns

Number - An integer indicating the total size, in bytes, of the movie clip.

Example

The following example uses the `_framesloaded` property to start a SWF file when all the frames are loaded. If all the frames aren't loaded, the `_xscale` property of the movie clip instance `loader` is increased proportionally to create a progress bar.

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

getBytesLoaded (MovieClip.getBytesLoaded method)

getDepth (MovieClip.getDepth method)

```
public getDepth() : Number
```

Returns the depth of the movie clip instance.

Each movie clip, button, and text field has a unique depth associated with it that determines how the object appears in front of or in back of other objects. Objects with higher depths appear in front.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Returns
Number - The depth of the movie clip.

Example
The following code traces the depth of all movie clip instances on the Stage:

```
for (var i in this) {
    if (typeof (this[i]) == "movieclip") {
    trace("movie clip '"+this[i]._name+"' is at depth "+this[i].getDepth());
    }
}
```

See also
`getInstanceAtDepth (MovieClip.getInstanceAtDepth method)`, `getNextHighestDepth (MovieClip.getNextHighestDepth method)`, `swapDepths (MovieClip.swapDepths method)`, `getDepth (TextField.getDepth method)`, `getDepth (Button.getDepth method)`

getInstanceAtDepth (MovieClip.getInstanceAtDepth method)

```
public getInstanceAtDepth(depth:Number) : MovieClip
```

Determines if a particular depth is already occupied by a movie clip. You can use this method before using `MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()`, or `MovieClip.createEmptyMovieClip()` to determine if the depth parameter you want to pass to any of these methods already contains a movie clip.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Parameters
depth : Number - An integer that specifies the depth level to query.

Returns
MovieClip - A reference to the MovieClip instance located at the specified depth, or `undefined` if there is no movie clip at that depth.

Example
The following example displays the depth occupied by the `triangle` movie clip instance in the Output panel:

```
this.createEmptyMovieClip("triangle", 1);

triangle.beginFill(0x0000FF, 100);
triangle.moveTo(100, 100);
triangle.lineTo(100, 150);
triangle.lineTo(150, 100);
triangle.lineTo(100, 100);

trace(this.getInstanceAtDepth(1)); // output: _level0.triangle
```

See also

attachMovie (MovieClip.attachMovie method), duplicateMovieClip (MovieClip.duplicateMovieClip method), createEmptyMovieClip (MovieClip.createEmptyMovieClip method), getDepth (MovieClip.getDepth method), getNextHighestDepth (MovieClip.getNextHighestDepth method), swapDepths (MovieClip.swapDepths method)

getNextHighestDepth (MovieClip.getNextHighestDepth method)

```
public getNextHighestDepth() : Number
```

Determines a depth value that you can pass to `MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()`, or `MovieClip.createEmptyMovieClip()` to ensure that Flash renders the movie clip in front of all other objects on the same level and layer in the current movie clip. The value returned is 0 or higher (that is, negative numbers are not returned).

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Returns

Number - An integer that reflects the next available depth index that would render above all other objects on the same level and layer within the movie clip.

Example

The following example draws three movie clip instances, using the `getNextHighestDepth()` method as the `depth` parameter of the `createEmptyMovieClip()` method, and labels each movie clip them with its depth:

```
for (i = 0; i < 3; i++) {
    drawClip(i);
}

function drawClip(n:Number):Void {
    this.createEmptyMovieClip("triangle" + n, this.getNextHighestDepth());
    var mc:MovieClip = eval("triangle" + n);
    mc.beginFill(0x00aaFF, 100);
    mc.lineStyle(4, 0xFF0000, 100);
    mc.moveTo(0, 0);
    mc.lineTo(100, 100);
    mc.lineTo(0, 100);
    mc.lineTo(0, 0);
    mc._x = n * 30;
    mc._y = n * 50
    mc.createTextField("label", this.getNextHighestDepth(), 20, 50, 200, 200)
    mc.label.text = mc.getDepth();
}
```

See also

getDepth (MovieClip.getDepth method), getInstanceAtDepth (MovieClip.getInstanceAtDepth method), swapDepths (MovieClip.swapDepths method), attachMovie (MovieClip.attachMovie method), duplicateMovieClip (MovieClip.duplicateMovieClip method), createEmptyMovieClip (MovieClip.createEmptyMovieClip method)

getSWFVersion (MovieClip.getSWFVersion method)

```
public getSWFVersion() : Number
```

Returns an integer that indicates the Flash Lite player version for the movie clip was published. If the movie clip is a JPEG, GIF, or PNG file, or if an error occurs and Flash can't determine the SWF version of the movie clip, -1 is returned.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Returns

Number - An integer that specifies the Flash Lite player version that was targeted when the SWF file loaded into the movie clip was published.

Example

The following example creates a new container and outputs the value of `getSWFVersion()`. It then uses MovieClipLoader to load an external SWF file that was published to Flash Player 7 and outputs the value of `getSWFVersion()` after the `onLoadInit` handler is triggered.

```
var container:MovieClip = this.createEmptyMovieClip("container", this.getUpperEmptyDepth());
var listener:Object = new Object();
listener.onLoadInit = function(target:MovieClip):Void {
    trace("target: " + target.getSWFVersion()); // target: 7
}
var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(listener);
trace("container: " + container.getSWFVersion()); // container: 8
mcLoader.loadClip("FlashPlayer7.swf", container);
```

getURL (MovieClip.getURL method)

```
public getURL(url:String, [window:String], [method:String]) : Void
```

Loads a document from the specified URL into the specified window. The `getURL()` method can also be used to pass variables to another application defined at the URL by using a `GET` or `POST` method.

Web pages that host Flash movies must explicitly set the `allowScriptAccess` attribute to allow or deny scripting for the Flash Lite player from the HTML code (in the `PARAM` tag for Internet Explorer or the `EMBED` tag for Netscape Navigator):

- When `allowScriptAccess` is `"never"`, outbound scripting always fails.
- When `allowScriptAccess` is `"always"`, outbound scripting always succeeds.
- When `allowScriptAccess` is `"sameDomain"` (supported by SWF files starting with version 8), outbound scripting is allowed if the SWF file is from the same domain as the hosting web page.
- If `allowScriptAccess` is not specified by an HTML page, it defaults to `"sameDomain"` for version 8 SWF files, and it defaults to `"always"` for earlier version SWF files.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

url : String - The URL from which to obtain the document.

window : String [optional] - A parameter specifying the name, frame, or expression that specifies the window or HTML frame that the document is loaded into. You can also use one of the following reserved target names: `_self` specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window.

method : String [optional] - A string (either `"GET"` or `"POST"`) that specifies a method for sending variables associated with the SWF file to load. If no variables are present, omit this parameter; otherwise, specify whether to load variables using a `GET` or `POST` method. `GET` appends the variables to the end of the URL and is used for a small number of variables. `POST` sends the variables in a separate HTTP header and is used for long strings of variables.

Example

The following ActionScript creates a new movie clip instance and opens the Adobe website in a new browser window:

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.getURL("http://www.adobe.com", "_blank");
```

The `getURL()` method also allows you to send variables to a remove server-side script, as seen in the following code:

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.username = "some user input";
loader_mc.password = "random string";
loader_mc.getURL("http://www.flash-mx.com/mm/viewscope.cfm", "_blank", "GET");
```

See also

getURL function, sendAndLoad (LoadVars.sendAndLoad method), send (LoadVars.send method)

globalToLocal (MovieClip.globalToLocal method)

```
public globalToLocal(pt:Object) : Void
```

Converts the *pt* object from Stage (global) coordinates to the movie clip's (local) coordinates.

The `MovieClip.globalToLocal()` method allows you to convert any given x and y coordinates from values that are relative to the top-left corner of the Stage to values that are relative to the top-left corner of a specific movie clip.

You must first create a generic object that has two properties, `x` and `y`. These x and y values (and they must be called `x` and `y`) are called the global coordinates because they relate to the top-left corner of the Stage. The `x` property represents the horizontal offset from the top-left corner. In other words, it represents how far to the right the point lies. For example, if x = 50, the point lies 50 pixels to the right of the top-left corner. The `y` property represents the vertical offset from the top-left corner. In other words, it represents how far down the point lies. For example, if `y` = 20, the point lies 20 pixels below the top-left corner. The following code creates a generic object with these coordinates:

```
var myPoint:Object = new Object();
myPoint.x = 50;
myPoint.y = 20;
```

Alternatively, you can create the object and assign the values at the same time with a literal Object value:

```
var myPoint:Object = {x:50, y:20};
```

After you create a point object with global coordinates, you can convert the coordinates to local coordinates. The `globalToLocal()` method doesn't return a value because it changes the values of x and y in the generic object that you send as the parameter. It changes them from values relative to the Stage (global coordinates) to values relative to a specific movie clip (local coordinates).

For example, if you create a movie clip that is positioned at the point `(_x:100, _y:100)`, and you pass the global point representing the top-left corner of the Stage `(x:0, y:0)` to the `globalToLocal()` method, the method should convert the x and y values to the local coordinates, which in this case is `(x:-100, y:-100)`. This is because the x and y coordinates are now expressed relative to the top-left corner of your movie clip rather than the top-left corner of the stage. The values are negative because to get from the top-left corner of your movie clip to the top-left corner of the Stage you have to move 100 pixels to the left (negative x) and 100 pixels up (negative y).

The movie clip coordinates were expressed using `_x` and `_y`, because those are the MovieClip properties that you use to set the x and y values for MovieClips. However, your generic object uses `x` and `y` without the underscore. The following code converts the *x* and *y* values to the local coordinates:

```
var myPoint:Object = {x:0, y:0}; // Create your generic point object.
this.createEmptyMovieClip("myMovieClip", this.getNextHighestDepth());
myMovieClip._x = 100; // _x for movieclip x position
myMovieClip._y = 100; // _y for movieclip y position

myMovieClip.globalToLocal(myPoint);
trace ("x: " + myPoint.x); // output: -100
trace ("y: " + myPoint.y); // output: -100
```

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Parameters
pt : Object - The name or identifier of an object created with the generic Object class. The object specifies the `x` and `y` coordinates as properties.

Example
Add the following ActionScript to a FLA or AS file in the same directory as an image called photo1.jpg:

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
coords_txt.html = true;
coords_txt.multiline = true;
coords_txt.autoSize = true;
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc._x = 100;
target_mc._y = 100;
target_mc.loadMovie("photo1.jpg");

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var point:Object = {x:_xmouse, y:_ymouse};
    target_mc.globalToLocal(point);
    var rowHeaders = "<b>   \t</b><b>_x\t</b><b>_y</b>";
    var row_1 = "_root\t"+_xmouse+"\t"+_ymouse;
    var row_2 = "target_mc\t"+point.x+"\t"+point.y;
    coords_txt.htmlText = "<textformat tabstops='[100, 150]'>";
    coords_txt.htmlText += rowHeaders;
    coords_txt.htmlText += row_1;
    coords_txt.htmlText += row_2;
    coords_txt.htmlText += "</textformat>";
};
Mouse.addListener(mouseListener);
```

See also
getBounds (MovieClip.getBounds method), localToGlobal (MovieClip.localToGlobal method), Object

gotoAndPlay (MovieClip.gotoAndPlay method)

```
public gotoAndPlay(frame:Object) : Void
```

Starts playing the SWF file at the specified frame. To specify a scene as well as a frame, use `gotoAndPlay()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Parameters
frame : Object - A number representing the frame number, or a string representing the label of the frame, to which the playhead is sent.

Example

The following example uses the `_framesloaded` property to start a SWF file when all of the frames are loaded. If all of the frames aren't loaded, the `_xscale` property of the `loader` movie clip instance is increased proportionally to create a progress bar.

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

`gotoAndPlay function`, `play function`

gotoAndStop (MovieClip.gotoAndStop method)

```
public gotoAndStop(frame:Object) : Void
```

Brings the playhead to the specified frame of the movie clip and stops it there. To specify a scene in addition to a frame, use `gotoAndStop()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

frame : Object - The frame number to which the playhead is sent.

Example

The following example uses the `_framesloaded` property to start a SWF file when all the frames are loaded. If all the frames aren't loaded, the `_xscale` property of the `loader` movie clip instance is increased proportionally to create a progress bar.

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

`gotoAndStop function`, `stop function`

_height (MovieClip._height property)

```
public _height : Number
```

The height of the movie clip, in pixels.

Availability

Flash Lite 2.0

Example

The following code example displays the height and width of a movie clip in the Output panel:

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var image_mcl:MovieClipLoader = new MovieClipLoader();
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace(target_mc._name+" = "+target_mc._width+" X "+target_mc._height+" pixels");
};
image_mcl.addListener(mclListener);

image_mcl.loadClip("example.jpg", image_mc);
```

See also

`_width (MovieClip._width property)`

_highquality (MovieClip._highquality property)

```
public _highquality : Number
```

Deprecated since Flash Player 7. This property was deprecated in favor of `MovieClip._quality`.

Specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the SWF file does not contain animation. Specify 0 (low quality) to prevent anti-aliasing. This property can overwrite the global `_highquality` property.

Availability

Flash Lite 2.0

Example

The following ActionScript specifies that best quality anti-aliasing should be applied to the SWF file.

```
    my_mc._highquality = 2;
```

See also

_quality (MovieClip._quality property), _quality property

hitArea (MovieClip.hitArea property)

```
public hitArea : Object
```

Designates another movie clip to serve as the hit area for a movie clip. If the `hitArea` property does not exist or is `null` or `undefined`, the movie clip itself is used as the hit area. The value of the `hitArea` property may be a reference to a movie clip object.

You can change the `hitArea` property at any time; the modified movie clip immediately takes on the new hit area behavior. The movie clip designated as the hit area does not need to be visible; its graphical shape, although not visible, is hit-tested. The `hitArea` property can be read out of a prototype object.

Availability

Flash Lite 2.0

Example

The following example sets the `circle_mc` movie clip as the hit area for the `square_mc` movie clip. Place these two movie clips on the Stage and test the document. When you click `circle_mc`, the `square_mc` movie clip traces that it was clicked.

```
square_mc.hitArea = circle_mc;
square_mc.onRelease = function() {
    trace("hit! "+this._name);
};
```

You can also set the `circle_mc` movie clip `visible` property to `false` to hide the hit area for `square_mc`.

```
circle_mc._visible = false;
```

See also

hitTest (MovieClip.hitTest method)

hitTest (MovieClip.hitTest method)

```
public hitTest() : Boolean
```

Evaluates the movie clip to see if it overlaps or intersects with the hit area that the `target` or `x` and `y` coordinate parameters identify.

Usage 1: Compares the *x* and *y* coordinates to the shape or bounding box of the specified instance, according to the `shapeFlag` setting. If `shapeFlag` is set to `true`, only the area actually occupied by the instance on the Stage is evaluated, and if *x* and *y* overlap at any point, a value of `true` is returned. This evaluation is useful for determining if the movie clip is within a specified hit or hotspot area.

Usage 2: Evaluates the bounding boxes of the `target` and specified instance, and returns `true` if they overlap or intersect at any point.

Parameters

`x: Number` The x coordinate of the hit area on the Stage.

`y: Number` The y coordinate of the hit area on the Stage.

The x and y coordinates are defined in the global coordinate space.

`shapeFlag: Boolean` A Boolean value specifying whether to evaluate the entire shape of the specified instance (`true`), or just the bounding box (`false`). This parameter can be specified only if the hit area is identified by using `x` and `y` coordinate parameters.

`target: Object` The target path of the hit area that may intersect or overlap with the movie clip. The `target` parameter usually represents a button or text-entry field.

Availability
Flash Lite 2.0

Returns
Boolean - A Boolean value of `true` if the movie clip overlaps with the specified hit area, `false` otherwise.

Example
The following example uses `hitTest()` to determine if the `circle_mc` movie clip overlaps or intersects the `square_mc` movie clip when the user releases the mouse button:

```
square_mc.onPress = function() {
    this.startDrag();
};
square_mc.onRelease = function() {
    this.stopDrag();
    if (this.hitTest(circle_mc)) {
    trace("you hit the circle");
    }
};
```

See also
`getBounds (MovieClip.getBounds method)`,`globalToLocal (MovieClip.globalToLocal method)`, `localToGlobal (MovieClip.localToGlobal method)`

lineStyle (MovieClip.lineStyle method)

```
public lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean,
noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number) : Void
```

Specifies a line style that Flash uses for subsequent calls to `lineTo()` and `curveTo()` until you call `lineStyle()` with different parameters. You can call `lineStyle()` in the middle of drawing a path to specify different styles for different line segments within a path.

Note: Calls to `clear()` set the line style back to `undefined`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Parameters

thickness : Number - An integer that indicates the thickness of the line in points; valid values are 0 to 255. If a number is not specified, or if the parameter is `undefined`, a line is not drawn. If a value of less than 0 is passed, Flash uses 0. The value 0 indicates hairline thickness; the maximum thickness is 255. If a value greater than 255 is passed, the Flash interpreter uses 255.

rgb : Number - A hex color value (for example, red is 0xFF0000, blue is 0x0000FF, and so on) of the line. If a value isn't indicated, Flash uses 0x000000 (black).

alpha : Number - An integer that indicates the alpha value of the line's color; valid values are 0 to 100. If a value isn't indicated, Flash uses 100 (solid). If the value is less than 0, Flash uses 0; if the value is greater than 100, Flash uses 100.

pixelHinting : Boolean - A Boolean value that specifies whether to hint strokes to full pixels. This affects both the position of anchors of a curve and the line stroke size itself. With pixelHinting set to true, Flash Lite player hints line widths to full pixel widths. With pixelHinting set to false, disjoints can appear for curves and straight lines.

noScale : String - A string that specifies how to scale a stroke. Valid values are as follows:

"normal"- Always scale the thickness (the default)."none"- Never scale the thickness."vertical"- Do not scale thickness if object is scaled vertically only."horizontal"- Do not scale thickness if object is scaled horizontally only

capsStyle : String - A string that specifies the type of caps at the end of lines. Valid values are: "round", "square", and "none". If a value is not indicated, Flash uses round caps.

jointStyle : String - A string that specifies the type of joint appearance used at angles. Valid values are: "round", "miter", and "bevel". If a value is not indicated, Flash uses round joints.

miterLimit : Number - A number that indicates the limit at which a miter is cut off. Valid values range from 1 to 255 (and values outside of that range are rounded to 1 or 255). This value is only used if the jointStyle is set to "miter". If a value is not indicated, Flash uses 3. The miterLimit value represents the length that a miter can extend beyond the point at which the lines meet to form a joint. The value expresses a factor of the line thickness. For example, with a miterLimit factor of 2.5 and a thickness of 10 pixels, the miter is cut off at 25 pixels.

Example

The following code draws a triangle with a 5-pixel, solid magenta line with no fill.

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.lineStyle(5, 0xff00ff, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
```

See also

beginFill (MovieClip.beginFill method), beginGradientFill (MovieClip.beginGradientFill method), clear (MovieClip.clear method), curveTo (MovieClip.curveTo method), lineTo (MovieClip.lineTo method), moveTo (MovieClip.moveTo method)

lineTo (MovieClip.lineTo method)

```
public lineTo(x:Number, y:Number) : Void
```

Draws a line using the current line style from the current drawing position to (*x*, *y*); the current drawing position is then set to (*x*, *y*). If the movie clip that you are drawing in contains content that was created with the Flash drawing tools, calls to `lineTo()` are drawn underneath the content. If you call `lineTo()` before any calls to the `moveTo()` method, the current drawing position defaults to (0,0). If any of the parameters are missing, this method fails and the current drawing position is not changed.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

x: Number - An integer indicating the horizontal position relative to the registration point of the parent movie clip.

y: Number - An integer indicating the vertical position relative to the registration point of the parent movie clip.

Example

The following example draws a triangle with a 5-pixel, solid magenta line and a partially transparent blue fill:

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

See also

beginFill (MovieClip.beginFill method), createEmptyMovieClip (MovieClip.createEmptyMovieClip method), endFill (MovieClip.endFill method), lineStyle (MovieClip.lineStyle method), moveTo (MovieClip.moveTo method)

loadMovie (MovieClip.loadMovie method)

```
public loadMovie(url:String, [method:String]) : Void
```

Loads SWF or JPEG files into a movie clip in a SWF file playing in Flash Lite.

Tip: To monitor the progress of the download, use the `MovieClipLoader.loadClip()` method instead of the `loadMovie()` method.

The `loadMovie()` method lets you display several SWF files at once and switch between SWF files without loading another HTML document.

A SWF file or image loaded into a movie clip inherits the position, rotation, and scale properties of the movie clip. You can use the target path of the movie clip to target the loaded SWF file.

Call `loadMovie()` to load any image format that the device supports. For example, if the target device supports PNG files, the following code loads and displays a PNG file that resides on a web server:

```
loadMovie("http://www.adobe.com/image.png", "image_target");
```

To determine what image formats the target device supports, use the `System.capabilities.imageMIMETypes` property, which contains an array of supported image MIME types. The index of each element in the array is equal to each supported MIME type. For example, the following code determines whether a device supports PNG images before the device attempts to load an external PNG file:

```
if (System.capabilities.imageMIMETypes["image/png"]) {
    loadMovie("images/image.png", "mc_myPngImage");
}
```

Flash Lite limits to five the number of `loadMovie()` operations that an application can perform in a given frame. Flash Lite limits to ten the total `loadMovie()` operations at any one time. For example, suppose your application contains code on Frame 1 that loads six external JPEG images:

```
image1.loadMovie("image1.jpg");
image2.loadMovie("image2.jpg");
image3.loadMovie("image3.jpg");
image4.loadMovie("image4.jpg");
image5.loadMovie("image5.jpg");
image6.loadMovie("image6.jpg"); // Won't load
```

In this case, only the first five images (image1.jpg through image5.jpg) load. The last image (image6.jpg) does not load because the five connection limit is reached. One solution is to split the `loadMovie()` calls over multiple frames so that each frame contains a maximum of five `loadMovie()` calls.

When you call the `loadMovie()` method, set the `MovieClip._lockroot` property to `true` in the loader movie, as shown in the following code example. If you don't set `_lockroot` to `true` in the loader movie, any references to `_root` in the loaded movie point to the `_root` of the loader instead of the `_root` of the loaded movie.

```
myMovieClip._lockroot = true;
```

Use the `MovieClip.unloadMovie()` method to remove SWF files or images loaded with the `loadMovie()` method.

Use the `MovieClip.loadVariables()` method, the XML object, Flash Remoting, or shared object to keep the active SWF file and load new data into it.

Using event handlers with `MovieClip.loadMovie()` can be unpredictable. If you attach an event handler to a button by using `on()`, or if you create a dynamic handler by using an event handler method such as `MovieClip.onPress`, and then you call `loadMovie()`, the event handler does not remain after the new content is loaded. However, if you attach an event handler to a movie clip by using `onClipEvent()` or `on()`, and then call `loadMovie()` on that movie clip, the event handler remains after the new content is loaded.

Availability

Flash Lite 2.0

Parameters

url : String - The absolute or relative URL of the SWF or JPEG file to be loaded. A relative path must be relative to the SWF file at level 0. Absolute URLs must include the protocol reference, such as `http://` or `file:///`.

method : String [optional] - Specifies an HTTP method for sending or loading variables. The parameter must be the string `GET` or `POST`. If no variables are to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Example

The following example creates a new movie clip, then creates child inside of it and loads a PNG image into the child. This allows the parent to retain any instance values that were assigned prior to the call to `loadMovie`.

```
var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.onRelease = function():Void {
    trace(this.image._url); // http://www.w3.org/Icons/w3c_main.png
}
var image:MovieClip = mc.createEmptyMovieClip("image", mc.getNextHighestDepth());
image.loadMovie("http://www.w3.org/Icons/w3c_main.png");
```

See also

`_lockroot (MovieClip._lockroot property)`, `unloadMovie (MovieClip.unloadMovie method)`, `loadVariables (MovieClip.loadVariables method)`, `loadMovie (MovieClip.loadMovie method)`, `onPress (MovieClip.onPress handler)`, `MovieClipLoader`, `onClipEvent handler`, `Constants`, `loadMovieNum function`, `unloadMovie function`, `unloadMovieNum function`

loadVariables (MovieClip.loadVariables method)

```
public loadVariables(url:String, [method:String]) : Void
```

Reads data from an external file and sets the values for variables in the movie clip. The external file can be a text file that ColdFusion generates, a CGI script, an Active Server Page (ASP), a PHP script, or any other properly formatted text file. The file can contain any number of variables.

The `loadVariables` method can also be used to update variables in the active movie clip with new values.

The `loadVariables` method requires that the text of the URL be in the standard MIME format: *application/x-www-form-urlencoded* (CGI script format).

In SWF files running in a version earlier than Flash Player 7, `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the left-most component of a file's URL. For example, a SWF file at www.someDomain.com can load data from a source at store.someDomain.com because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `url` must be in exactly the same domain as the SWF file that is issuing this call. For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. To load data from a different domain, you can place a *cross-domain policy file* on the server hosting the data source that is being accessed.

To load variables into a specific level, use `loadVariablesNum()` instead of `loadVariables()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

url : String - The absolute or relative URL for the external file that contains the variables to be loaded. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see "Description," below.

method : String [optional] - Specifies an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If no variables are sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Example

The following example loads information from a text file called params.txtinto the `target_mc` movie clip that is created by using `createEmptyMovieClip()`. The `setInterval()` function is used to check the loading progress. The script checks for a variable in the params.txt file named `done`.

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc.loadVariables("params.txt");
function checkParamsLoaded() {
    if (target_mc.done == undefined) {
    trace("not yet.");
    } else {
    trace("finished loading. killing interval.");
    trace("-------------");
    for (i in target_mc) {
        trace(i+": "+target_mc[i]);
    }
    trace("-------------");
    clearInterval(param_interval);
    }
}
var param_interval = setInterval(checkParamsLoaded, 100);
```

The params.txt file includes the following text:

```
var1="hello"&var2="goodbye"&done="done"
```

See also

loadMovie (MovieClip.loadMovie method), loadVariablesNum function, unloadMovie (MovieClip.unloadMovie method)

localToGlobal (MovieClip.localToGlobal method)

```
public localToGlobal(pt:Object) : Void
```

Converts the *pt* object from the movie clip's (local) coordinates to the Stage (global) coordinates.

The `MovieClip.localToGlobal()` method allows you to convert any given x and y coordinates from values that are relative to the top-left corner of a specific movie clip to values that are relative to the top-left corner of the Stage.

You must first create a generic object that has two properties, `x` and `y`. These `x` and `y` values (and they must be called `x` and `y`) are called the local coordinates because they relate to the top-left corner of the movie clip. The `x` property represents the horizontal offset from the top-left corner of the movie clip. In other words, it represents how far to the right the point lies. For example, if `x` = 50, the point lies 50 pixels to the right of the top-left corner. The `y` property represents the vertical offset from the top-left corner of the movie clip. In other words, it represents how far down the point lies. For example, if `y` = 20, the point lies 20 pixels below the top-left corner. The following code creates a generic object with these coordinates.

```
var myPoint:Object = new Object();
myPoint.x = 50;
myPoint.y = 20;
```

Alternatively, you can create the object and assign the values at the same time with a literal Object value.

```
var myPoint:Object = {x:50, y:20};
```

After you create a point object with local coordinates, you can convert the coordinates to global coordinates. The `localToGlobal()` method doesn't return a value because it changes the values of `x` and `y` in the generic object that you send as the parameter. It changes them from values relative to a specific movie clip (local coordinates) to values relative to the Stage (global coordinates).

For example, if you create a movie clip that is positioned at the point (_x:100, _y:100), and you pass a local point representing a point near the top-left corner of the movie clip (x:10, y:10) to the `localToGlobal()` method, the method should convert the x and y values to global coordinates, which in this case is (x:110, y:110). This conversion occurs because the *x* and *y* coordinates are now expressed relative to the top-left corner of the Stage rather than the top-left corner of your movie clip.

The movie clip coordinates were expressed using `_x` and `_y`, because those are the MovieClip properties that you use to set the x and y values for MovieClips. However, your generic object uses `x` and `y` without the underscore. The following code converts the x and y coordinates to global coordinates:

```
var myPoint:Object = {x:10, y:10}; // create your generic point object
this.createEmptyMovieClip("myMovieClip", this.getNextHighestDepth());
myMovieClip._x = 100; // _x for movieclip x position
myMovieClip._y = 100; // _y for movieclip y position

myMovieClip.localToGlobal(myPoint);
trace ("x: " + myPoint.x); // output: 110
trace ("y: " + myPoint.y); // output: 110
```

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

pt : Object - The name or identifier of an object created with the Object class, specifying the *x* and *y* coordinates as properties.

Example

The following example converts `x` and `y` coordinates of the my_mc object, from the movie clip's (local) coordinates to the Stage (global) coordinates. The center point of the movie clip is reflected after you click and drag the instance.

```
this.createTextField("point_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var point:Object = {x:my_mc._width/2, y:my_mc._height/2};
    my_mc.localToGlobal(point);
    point_txt.text = "x:"+point.x+", y:"+point.y;
};
Mouse.addListener(mouseListener);
my_mc.onPress = function() {
    this.startDrag();
};
my_mc.onRelease = function() {
    this.stopDrag();
};
```

See also

globalToLocal (MovieClip.globalToLocal method)

_lockroot (MovieClip._lockroot property)

```
public _lockroot : Boolean
```

A Boolean value that specifies what `_root` refers to when a SWF file is loaded into a movie clip. The `_lockroot` property is `undefined` by default. You can set this property within the SWF file that is being loaded or in the handler that is loading the movie clip.

For example, suppose you have a document called Games.fla that lets a user choose a game to play, and loads the game (for example, Chess.swf) into the `game_mc` movie clip. Make sure that, after being loaded into Games.swf, any use of `_root` in Chess.swf will refer to `_root` in Chess.swf (not `_root` in Games.swf). If you have access to Chess.fla and publish it to Flash Player 7 or later, you can add this statement to Chess.fla on the main Timeline:

```
this._lockroot = true;
```

If you don't have access to Chess.fla (for example, if you are loading Chess.swf from someone else's site into `chess_mc`), you can set the Chess.swf `_lockroot` property when you load it. Place the following ActionScript on the main Timeline of Games.fla:

```
chess_mc._lockroot = true;
```

In this case, Chess.swf can be published for any version of Flash Player, as long as Games.swf is published for Flash Player 7 or later.

When calling `loadMovie()`, set the `MovieClip._lockroot` property to `true` in the loader movie, as shown in the following code. If you don't set `_lockroot` to `true` in the loader movie, any references to `_root` in the loaded movie point to the `_root` of the loader instead of the `_root` of the loaded movie:

```
myMovieClip._lockroot = true;
```

Availability

Flash Lite 2.0

Example

In the following example, lockroot.fla has `_lockroot` applied to the main SWF file. If the SWF file is loaded into another FLA document, `_root` always refers to the scope of lockroot.swf, which helps prevent conflicts. Place the following ActionScript on the main Timeline of lockroot.fla:

```
this._lockroot = true;
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from lockroot.swf");
for (i in _root) {
    trace(" "+i+" -> "+_root[i]);
}
trace("");
```

which traces the following information:

```
from lockroot.swf
myOtherVar -> 2
myVar -> 1
_lockroot -> true
$version -> WIN 7,0,19,0
```

The following example loads two SWF files, lockroot.swf and nolockroot.swf. The lockroot.fla document contains the ActionScript from the preceding example. The nolockroot FLA file has the following code placed on Frame 1 of the Timeline:

```
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from nolockroot.swf");
for (i in _root) {
    trace(" "+i+" -> "+_root[i]);
}
trace("");
```

The lockroot.swf file has `_lockroot` applied to it, and nolockroot.swf does not. After the files are loaded, each file dumps variables from their `_root` scopes. Place the following ActionScript on the main Timeline of a FLA document:

```
this.createEmptyMovieClip("lockroot_mc", this.getNextHighestDepth());
lockroot_mc.loadMovie("lockroot.swf");
this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
nolockroot_mc.loadMovie("nolockroot.swf");
function dumpRoot() {
    trace("from current SWF file");
    for (i in _root) {
    trace(" "+i+" -> "+_root[i]);
    }
    trace("");
}
dumpRoot();
```

which traces the following information:

```
from current SWF file
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myVar -> 1
i -> lockroot_mc
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from lockroot.swf
myOtherVar -> 2
myVar -> 1
```

The file with no `_lockroot` applied also contains all of the other variables that the root SWF file contains. If you don't have access to the nolockroot.fla, you can use the following ActionScript added to the main Timeline to change the `_lockroot` in the preceding main FLA document:

```
this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
nolockroot_mc._lockroot = true;
nolockroot_mc.loadMovie("nolockroot.swf");
```

which then traces the following:

```
from current SWF file
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myOtherVar -> 2
myVar -> 1

from lockroot.swf
myOtherVar -> 2
myVar -> 1
```

See also

_root property, _lockroot (MovieClip._lockroot property), attachMovie (MovieClip.attachMovie method), loadMovie (MovieClip.loadMovie method), onLoadInit (MovieClipLoader.onLoadInit event listener)

moveTo (MovieClip.moveTo method)

```
public moveTo(x:Number, y:Number) : Void
```

Moves the current drawing position to (*x*, *y*). If any of the parameters are missing, this method fails and the current drawing position is not changed.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

x : Number - An integer indicating the horizontal position relative to the registration point of the parent movie clip.

y : Number - An integer indicating the vertical position relative to the registration point of the parent movie clip.

Example

The following example draws a triangle with a 5-pixel, solid magenta line and a partially transparent blue fill:

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

See also

createEmptyMovieClip (MovieClip.createEmptyMovieClip method), lineStyle (MovieClip.lineStyle method), lineTo (MovieClip.lineTo method)

_name (MovieClip._name property)

```
public _name : String
```

The instance name of the movie clip.

Availability

Flash Lite 2.0

See also

_name (Button._name property)

nextFrame (MovieClip.nextFrame method)

```
public nextFrame() : Void
```

Sends the playhead to the next frame and stops it.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Example

The following example uses `_framesloaded` and `nextFrame()` to load content into a SWF file. Do not add any code to Frame 1, but add the following ActionScript to Frame 2 of the Timeline:

```
if (this._framesloaded >= 3) {
    this.nextFrame();
} else {
    this.gotoAndPlay(1);
}
```

Then, add the following code (and the content you want to load) on Frame 3:

```
stop();
```

See also

nextFrame function, prevFrame function, prevFrame (MovieClip.prevFrame method)

onData (MovieClip.onData handler)

```
onData = function() {}
```

Invoked when a movie clip receives data from a MovieClip.loadVariables() or MovieClip.loadMovie() call. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

This handler can be used only with movie clips for which you have a symbol in the library that is associated with a class. If you want an event handler to be invoked when a specific movie clip receives data, you must use onClipEvent() instead of this handler. The latter handler is invoked when any movie clip receives data.

Availability

Flash Lite 2.0

Example

The following example illustrates the correct use of `MovieClip.onData()` and `onClipEvent(data)`.

The symbol_mc is a movie clip symbol in the library. It is linked to the MovieClip class. The first function below is triggered for each instance of symbol_mc when it receives data.

The dynamic_mc is a movie clip that is being loaded with MovieClip.loadMovie(). The code using dynamic_mc below attempts to call a function when the movie clip is loaded, but it doesn't work. The loaded SWF file must be a symbol in the library associated with the MovieClip class.

The last function uses onClipEvent(data). The onClipEvent() event handler is invoked for any movie clip that receives data, whether the movie clip is in the library or not. Therefore, the last function in this example is invoked when symbol_mc is instantiated and also when replacement.swf is loaded.

```
// The following function is triggered for each instance of symbol_mc
// when it receives data.
symbol_mc.onData = function() {
    trace("The movie clip has received data");
}

// This code attempts to call a function when the clip is loaded,
// but it will not work, because the loaded SWF is not a symbol
// in the library associated with the MovieClip class.
function output()
{
    trace("Will never be called.");
}
dynamic_mc.onData = output;
dynamic_mc.loadMovie("replacement.swf");
// The following function is invoked for any movie clip that
// receives data, whether it is in the library or not.
onClipEvent( data ) {
    trace("The movie clip has received data");
}
```

See also

`onClipEvent handler`

onDragOut (MovieClip.onDragOut handler)

```
onDragOut = function() {}
```

Invoked when the mouse button is pressed and the pointer rolls outside the object. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Note: This event handler is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example defines a function for the `onDragOut` method that sends a `trace()` action to the Output panel:

```
my_mc.onDragOut = function () {
    trace ("onDragOut called");
}
```

See also

onDragOver (MovieClip.onDragOver handler)

onDragOver (MovieClip.onDragOver handler)

```
onDragOver = function() {}
```

Invoked when the pointer is dragged outside and then over the movie clip. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Note: This event handler is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example defines a function for the `onDragOver` method that sends a `trace()` action to the Output panel:

```
my_mc.onDragOver = function () {
    trace ("onDragOver called");
}
```

See also

onDragOut (MovieClip.onDragOut handler)

onEnterFrame (MovieClip.onEnterFrame handler)

```
onEnterFrame = function() {}
```

Invoked repeatedly at the frame rate of the SWF file. The function that you assign to the `onEnterFrame` event handler is processed before any other ActionScript code that is attached to the affected frames.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or that is linked to a symbol in the library.

Availability

Flash Lite 2.0

Example

The following example defines a function for the `onEnterFrame` event handler that sends a `trace()` action to the Output panel:

```
my_mc.onEnterFrame = function () {
    trace ("onEnterFrame called");
}
```

onKeyDown (MovieClip.onKeyDown handler)

```
onKeyDown = function() {}
```

Invoked when a movie clip has input focus and a key is pressed. The `onKeyDown` event handler is invoked with no parameters. You can use the `Key.getAscii()` and `Key.getCode()` methods to determine which key was pressed. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

The `onKeyDown` event handler works only if the movie clip has input focus enabled and set. First, the `MovieClip.focusEnabled` property must be set to `true` for the movie clip. Then, the clip must be given focus. This can be done either by using `Selection.setFocus()` or by setting the Tab key to navigate to the clip.

If `Selection.setFocus()` is used, the path for the movie clip must be passed to `Selection.setFocus()`. It is very easy for other elements to take the focus back after the mouse is moved.

Availability

Flash Lite 2.0

Example

The following example defines a function for the `onKeyDown()` method that sends a `trace()` action to the Output panel. Create a movie clip called my_mc and add the following ActionScript to your FLA or AS file:

```
my_mc.onKeyDown = function () {
    trace ("key was pressed");
}
```

The movie clip must have focus for the onKeyDown event handler to work. Add the following ActionScript to set input focus:

```
my_mc.tabEnabled = true;
my_mc.focusEnabled = true;
Selection.setFocus(my_mc);
```

When you tab to the movie clip and press a key, `key was pressed` is displayed in the Output panel. However, this does not occur after you move the mouse, because the movie clip loses focus. Therefore, you should use `Key.onKeyDown` in most cases.

See also

`getAscii (Key.getAscii method)`, `getCode (Key.getCode method)`, `focusEnabled (MovieClip.focusEnabled property)`, `setFocus (Selection.setFocus method)`, `onKeyDown (Key.onKeyDown event listener)`, `onKeyUp (MovieClip.onKeyUp handler)`

onKeyUp (MovieClip.onKeyUp handler)

```
onKeyUp = function() {}
```

Invoked when a key is released. The `onKeyUp` event handler is invoked with no parameters. You can use the `Key.getAscii()` and `Key.getCode()` methods to determine which key was pressed. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

The `onKeyUp` event handler works only if the movie clip has input focus enabled and set. First, the `MovieClip.focusEnabled` property must be set to true for the movie clip. Then, the clip must be given focus. This can be done either by using `Selection.setFocus()` or by setting the Tab key to navigate to the clip.

If `Selection.setFocus()` is used, the path for the movie clip must be passed to `Selection.setFocus()`. It is very easy for other elements to take the focus back after the mouse is moved.

Availability
Flash Lite 2.0

Example
The following example defines a function for the `onKeyUp` method that sends a `trace()` action to the Output panel:

```
my_mc.onKeyUp = function () {
    trace ("onKey called");
}
```

The following example sets input focus:

```
my_mc.focusEnabled = true;
Selection.setFocus(my_mc);
```

See also
`getAscii (Key.getAscii method)`, `getCode (Key.getCode method)`, `focusEnabled (MovieClip.focusEnabled property)`, `setFocus (Selection.setFocus method)`, `onKeyDown (Key.onKeyDown event listener)`, `onKeyDown (MovieClip.onKeyDown handler)`

onKillFocus (MovieClip.onKillFocus handler)

```
onKillFocus = function(newFocus:Object) {}
```

Invoked when a movie clip loses input focus. The `onKillFocus` method receives one parameter, `newFocus`, which is an object that represents the new object receiving the focus. If no object receives the focus, `newFocus` contains the value `null`.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Availability
Flash Lite 2.0

Parameters
newFocus : Object - The object that is receiving the input focus.

Example
The following example displays information about the movie clip that loses focus, and the instance that currently has focus. Two movie clips, called `my_mc` and `other_mc`, are on the Stage. You can add the following ActionScript to your AS or FLA document:

```
my_mc.onRelease = Void;
other_mc.onRelease = Void;
my_mc.onKillFocus = function(newFocus) {
    trace("onKillFocus called, new focus is: "+newFocus);
};
```

When you press the Tab key to move between the two instances, information is displayed in the Output panel.

See also

onSetFocus (MovieClip.onSetFocus handler)

onLoad (MovieClip.onLoad handler)

```
onLoad = function() {}
```

Invoked when the movie clip is instantiated and appears in the Timeline. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

This handler can be used only with movie clips for which you have a symbol in the library that is associated with a class. If you want an event handler to be invoked when a specific movie clip loads, for example when you use `MovieClip.loadMovie()` to load a SWF file dynamically, you must use onClipEvent(load) or the MovieClipLoader class instead of this handler. Unlike `MovieClip.onLoad`, the other handlers are invoked when any movie clip loads.

Availability

Flash Lite 2.0

Example

This example shows you how to use the `onLoad` event handler in an ActionScript 2.0 class definition that extends the MovieClip class. First, create a class file named Oval.as and define a class method named `onLoad()` and make sure that the class file is placed in the proper class path:

```
// contents of Oval.as
class Oval extends MovieClip{
    public function onLoad () {
        trace ("onLoad called");
    }
}
```

Second, create a movie clip symbol in your library and name it Oval. Context-click (usually right-click) on the symbol in the Library panel and select Linkage... from the pop-up menu. Click on "Export for ActionScript" and fill in the "Identifier" and "ActionScript 2.0 Class" fields with the word "Oval" (no quotes). Leave "Export in First Frame" checked and click OK.

Third, go to the first frame of your file and enter the following code in the Actions Panel:

```
var myOval:Oval = Oval(attachMovie("Oval","Oval_1",1));
```

Finally, do a test movie, and you should see the output text "onLoad called".

See also

loadMovie (MovieClip.loadMovie method), onClipEvent handler, MovieClipLoader

onMouseDown (MovieClip.onMouseDown handler)

```
onMouseDown = function() {}
```

Invoked when the mouse button is pressed. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Note: This event handler is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
The following example defines a function for the `onMouseDown` method that sends a `trace()` action to the Output panel:

```
my_mc.onMouseDown = function () {
    trace ("onMouseDown called");
}
```

onMouseMove (MovieClip.onMouseMove handler)

```
onMouseMove = function() {}
```

Invoked when the mouse moves. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Note: This event handler is supported in Flash Lite only if `System.capabilities.hasMouse` is `true`.

Availability
Flash Lite 2.0

Example
The following example defines a function for the `onMouseMove` method that sends a `trace()` action to the Output panel:

```
my_mc.onMouseMove = function () {
    trace ("onMouseMove called");
}
```

onMouseUp (MovieClip.onMouseUp handler)

```
onMouseUp = function() {}
```

Invoked when the mouse button is released. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Note: This event handler is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
The following example defines a function for the `onMouseUp` method that sends a `trace()` action to the Output panel:

```
my_mc.onMouseUp = function () {
    trace ("onMouseUp called");
}
```

onPress (MovieClip.onPress handler)

```
onPress = function() {}
```

Invoked when the user clicks the mouse while the pointer is over a movie clip. You must define a function that executes when the event handler is invoked. You can define the in the library.

Availability

Flash Lite 2.0

Example

The following example defines a function for the `onPress` method that sends a `trace()` action to the Output panel:

```
my_mc.onPress = function () {
    trace ("onPress called");
}
```

onRelease (MovieClip.onRelease handler)

```
onRelease = function() {}
```

Invoked when the mouse button is released over a movie clip. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Availability

Flash Lite 2.0

Example

The following example defines a function for the `onRelease` method that sends a `trace()` action to the Output panel:

```
my_mc.onRelease = function () {
    trace ("onRelease called");
}
```

onReleaseOutside (MovieClip.onReleaseOutside handler)

```
onReleaseOutside = function() {}
```

Invoked when the mouse button is pressed inside the movie clip area and then released outside the movie clip area.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Note: This event handler is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example defines a function for the `onReleaseOutside` method that sends a `trace()` action to the Output panel:

```
my_mc.onReleaseOutside = function () {
    trace ("onReleaseOutside called");
}
```

onRollOut (MovieClip.onRollOut handler)

```
onRollOut = function() {}
```

Invoked when the pointer moves outside a movie clip area.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Availability
Flash Lite 2.0

Example
The following example defines a function for the `onRollOut` method that sends a `trace()` action to the Output panel:

```
my_mc.onRollOut = function () {
    trace ("onRollOut called");
}
```

onRollOver (MovieClip.onRollOver handler)

```
onRollOver = function() {}
```

Invoked when the pointer moves over a movie clip area.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Availability
Flash Lite 2.0

Example
The following example defines a function for the `onRollOver` method that sends a `trace()` action to the Output panel:

```
my_mc.onRollOver = function () {
    trace ("onRollOver called");
}
```

onSetFocus (MovieClip.onSetFocus handler)

```
onSetFocus = function(oldFocus:Object) {}
```

Invoked when a movie clip receives input focus. The `oldFocus` parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a movie clip to a text field, `oldFocus` contains the movie clip instance.

If there is no previously focused object, `oldFocus` contains a null value.

You must define a function that executes when the event handler in invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Availability
Flash Lite 2.0

Parameters
oldFocus : Object - The object to lose focus.

Example
The following example displays information about the movie clip that receives input focus, and the instance that previously had focus. Two movie clips, called `my_mc` and `other_mc` are on the Stage. Add the following ActionScript to your AS or FLA document:

```
my_mc.onRelease = Void;
other_mc.onRelease = Void;
my_mc.onSetFocus = function(oldFocus) {
    trace("onSetFocus called, previous focus was: "+oldFocus);
}
```

When you press the Tab key between the two instances, information is displayed in the Output panel.

See also
`onKillFocus (MovieClip.onKillFocus handler)`

onUnload (MovieClip.onUnload handler)

```
onUnload = function() {}
```

Invoked in the first frame after the movie clip is removed from the Timeline. Flash processes the actions associated with the `onUnload` event handler before attaching any actions to the affected frame. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library.

Availability
Flash Lite 2.0

Example
The following example defines a function for the `MovieClip.onUnload` method that sends a `trace()` action to the Output panel:

```
my_mc.onUnload = function () {
    trace ("onUnload called");
}
```

_parent (MovieClip._parent property)

```
public _parent : MovieClip
```

A reference to the movie clip or object that contains the current movie clip or object. The current object is the object that references the `_parent` property. Use the `_parent` property to specify a relative path to movie clips or objects that are above the current movie clip or object.

You can use `_parent` to move up multiple levels in the display list as in the following:

```
this._parent._parent._alpha = 20;
```

Availability
Flash Lite 2.0

Example
The following example traces the reference to a movie clip and its relationship to the main Timeline. Create a movie clip with the instance name `my_mc`, and add it to the main Timeline. Add the following ActionScript to your FLA or AS file:

```
my_mc.onRelease = function() {
    trace("You clicked the movie clip: "+this);
    trace("The parent of "+this._name+" is: "+this._parent);
}
```

When you click the movie clip, the following information appears in the Output panel:

```
You clicked the movie clip: _level0.my_mc
The parent of my_mc is: _level0
```

See also
`_parent (Button._parent property)`, `_root property`, `targetPath function`, `_parent (TextField._parent property)`

play (MovieClip.play method)

```
public play() : Void
```

Moves the playhead in the Timeline of the movie clip.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Example
Use the following ActionScript to play the main Timeline of a SWF file. This ActionScript is for a movie clip button called `my_mc` on the main Timeline:

```
stop();
my_mc.onRelease = function() {
    this._parent.play();
};
```

Use the following ActionScript to play the Timeline of a movie clip in a SWF file. This ActionScript is for a button called `my_btn` on the main Timeline that plays a movie clip called `animation_mc`:

```
animation_mc.stop();
my_btn.onRelease = function(){
    animation_mc.play();
};
```

See also
`play function`, `gotoAndPlay (MovieClip.gotoAndPlay method)`, `gotoAndPlay function`

prevFrame (MovieClip.prevFrame method)

```
public prevFrame() : Void
```

Sends the playhead to the previous frame and stops it.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Example

In the following example, two movie clip buttons control the Timeline. The `prev_mc` button moves the playhead to the previous frame, and the `next_mc` button moves the playhead to the next frame. Add content to a series of frames on the Timeline, and add the following ActionScript to Frame 1 of the Timeline:

```
stop();
prev_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe>1) {
    parent_mc.prevFrame();
    } else {
    parent_mc.gotoAndStop(parent_mc._totalframes);
    }
};
next_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe<parent_mc._totalframes) {
    parent_mc.nextFrame();
    } else {
    parent_mc.gotoAndStop(1);
    }
};
```

See also

`prevFrame function`

_quality (MovieClip._quality property)

```
public _quality : String
```

Sets or retrieves the rendering quality used for a SWF file. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

Value	Description	Graphic Anti-Aliasing
`"LOW"`	Low rendering quality.	Graphics are not anti-aliased.

Value	Description	Graphic Anti-Aliasing
"MEDIUM"	Medium rendering quality. This setting is suitable for movies that do not contain text.	Graphics are anti-aliased using a 2 x 2 pixel grid.
"HIGH"	High rendering quality. This setting is the default rendering quality setting that Flash uses.	Graphics are anti-aliased using a 4 x 4 pixel grid.
"BEST"	Very high rendering quality.	Graphics are anti-aliased using a 4 x 4 pixel grid.

Note: Although you can specify this property for a MovieClip object, it is also a global property, and you can specify its value simply as `_quality`.

Availability

Flash Lite 2.0

Example

This example sets the rendering quality of a movie clip named `my_mc` to `LOW`:

```
my_mc._quality = "LOW";
```

See also

`_quality property`

removeMovieClip (MovieClip.removeMovieClip method)

```
public removeMovieClip() : Void
```

Removes a movie clip instance created with `duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`, `MovieClip.createEmptyMovieClip()`, or `MovieClip.attachMovie()`.

This method does not remove a movie clip assigned to a negative depth value. Movie clips created in the authoring tool are assigned negative depth values by default. To remove a movie clip that is assigned to a negative depth value, first use `MovieClip.swapDepths()` to move the movie clip to a positive depth value.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Example

Each time you click a button in the following example, you attach a movie clip instance to the Stage in a random position. When you click a movie clip instance, you remove that instance from the SWF file.

```
function randRange(min:Number, max:Number):Number {
    var randNum:Number = Math.round(Math.random()*(max-min))+min;
    return randNum;
}
var bugNum:Number = 0;
addBug_btn.onRelease = addBug;
function addBug() {
    var thisBug:MovieClip = this._parent.attachMovie("bug_id", "bug"+bugNum+"_mc", bugNum,
{_x:randRange(50, 500), _y:randRange(50, 350)});
    thisBug.onRelease = function() {
    this.removeMovieClip();
    };
    bugNum++;
}
```

See also

duplicateMovieClip function, createEmptyMovieClip (MovieClip.createEmptyMovieClip method), duplicateMovieClip (MovieClip.duplicateMovieClip method), attachMovie (MovieClip.attachMovie method) swapDepths (MovieClip.swapDepths method)

_rotation (MovieClip._rotation property)

```
public _rotation : Number
```

Specifies the rotation of the movie clip, in degrees, from its original orientation. Values from 0 to 180 represent clockwise rotation; values from 0 to -180 represent counterclockwise rotation. Values outside this range are added to or subtracted from 360 to obtain a value within the range. For example, the statement `my_mc._rotation = 450` is the same as `my_mc._rotation = 90`.

Availability

Flash Lite 2.0

Example

The following example creates a `triangle` movie clip instance dynamically. When you run the SWF file, click the movie clip to rotate it:

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());

triangle.beginFill(0x0000FF, 100);
triangle.moveTo(100, 100);
triangle.lineTo(100, 150);
triangle.lineTo(150, 100);
triangle.lineTo(100, 100);

triangle.onMouseUp= function() {
    this._rotation += 15;
};
```

See also

_rotation (Button._rotation property), _rotation (TextField._rotation property)

setMask (MovieClip.setMask method)

```
public setMask(mc:Object) : Void
```

Makes the movie clip in the parameter *mc* a mask that reveals the calling movie clip.

The `setMask()` method allows multiple-frame movie clips with complex, multilayered content to act as masks (which is possible by using mask layers). If you have device fonts in a masked movie clip, they are drawn but not masked. You can't set a movie clip to be its own maskfor example, `my_mc.setMask(my_mc)`.

If you create a mask layer that contains a movie clip, and then apply the `setMask()` method to it, the `setMask()` call takes priority and this is not reversible. For example, you could have a movie clip in a mask layer called `UIMask` that masks another layer that contains another movie clip called `UIMaskee`. If, as the SWF file plays, you call `UIMask.setMask(UIMaskee)`, from that point on, `UIMask` is masked by `UIMaskee`.

To cancel a mask created with ActionScript, pass the value `null` to the `setMask()` method. The following code cancels the mask without affecting the mask layer in the Timeline.

```
UIMask.setMask(null);
```

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Parameters
mc : Object - The instance name of a movie clip to be a mask. This can be a String or a MovieClip.

Example
The following code uses the `circleMask_mc` movie clip to mask the `theMaskee_mc` movie clip:

```
theMaskee_mc.setMask(circleMask_mc);
```

_soundbuftime (MovieClip._soundbuftime property)

```
public _soundbuftime : Number
```

Specifies the number of seconds a sound prebuffers before it starts to stream.

Note: Although you can specify this property for a MovieClip object, it is actually a global property that applies to all sounds loaded, and you can specify its value simply as `_soundbuftime`. Setting this property for a MovieClip object actually sets the global property.

Availability
Flash Lite 2.0

See also
`_soundbuftime property`

startDrag (MovieClip.startDrag method)

```
public startDrag([lockCenter:Boolean], [left:Number], [top:Number], [right:Number],
[bottom:Number]) : Void
```

Lets the user drag the specified movie clip. The movie clip remains draggable until explicitly stopped through a call to `MovieClip.stopDrag()`, or until another movie clip is made draggable. Only one movie clip at a time is draggable.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Note: This method is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Parameters

lockCenter : Boolean [optional] - A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (`true`), or locked to the point where the user first clicked on the movie clip (`false`).

left : Number [optional] - Value relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip.

top : Number [optional] - Value relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip.

right : Number [optional] - Value relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip.

bottom : Number [optional] - Value relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip.

Example

The following example creates a draggable movie clip instance called `mc_1`:

```
this.createEmptyMovieClip("mc_1", 1);

with (mc_1) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0x4827CF);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
}

mc_1.onPress = function() {
    this.startDrag();
};
mc_1.onRelease = function() {
    this.stopDrag();
};
```

See also

`_droptarget (MovieClip._droptarget property)`, `startDrag function`, `stopDrag (MovieClip.stopDrag method)`

stop (MovieClip.stop method)

```
public stop() : Void
```

Stops the movie clip currently playing.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Example
The following example shows how to stop a movie clip named `aMovieClip`:

```
aMovieClip.stop();
```

See also
`stop function`

stopDrag (MovieClip.stopDrag method)

```
public stopDrag() : Void
```

Ends a `MovieClip.startDrag()` method. A movie clip that was made draggable with that method remains draggable until a `stopDrag()` method is added, or until another movie clip becomes draggable. Only one movie clip is draggable at a time.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Note: This method is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
The following example creates a draggable movie clip instance called `mc_1`:

```
this.createEmptyMovieClip("mc_1", 1);

with (mc_1) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0x4827CF);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
}

mc_1.onPress = function() {
    this.startDrag();
};
mc_1.onRelease = function() {
    this.stopDrag();
};
```

See also

`_droptarget (MovieClip._droptarget property)`, `startDrag (MovieClip.startDrag method)`, `stopDrag function`

swapDepths (MovieClip.swapDepths method)

```
public swapDepths(target:Object) : Void
```

Swaps the stacking, or depth level (z-order), of this movie clip with the movie clip specified by the `target` parameter, or with the movie clip that currently occupies the depth level specified in the `target` parameter. Both movie clips must have the same parent movie clip. Swapping the depth level of movie clips has the effect of moving one movie clip in front of or behind the other. If a movie clip is tweening when this method is called, the tweening is stopped.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability

Flash Lite 2.0

Parameters

target : Object - This parameter can take one of two forms:

- A number that specifies the depth level where the movie clip is to be placed.
- A string that specifies the movie clip instance whose depth is swapped with the movie clip for which the method is being applied. Both movie clips must have the same parent movie clip.

Example

The following example swaps the stacking order of two movie clip instances. Overlap two movie clip instances, called `myMC1_mc` and `myMC2_mc`, on the Stage and then add the following script to the parent Timeline:

```
myMC1_mc.onRelease = function() {
    this.swapDepths(myMC2_mc);
};
myMC2_mc.onRelease = function() {
    this.swapDepths(myMC1_mc);
};
```

See also

_level property, getDepth (MovieClip.getDepth method), getInstanceAtDepth (MovieClip.getInstanceAtDepth method), getNextHighestDepth (MovieClip.getNextHighestDepth method)

tabChildren (MovieClip.tabChildren property)

```
public tabChildren : Boolean
```

Determines whether the children of a movie clip are included in the automatic tab ordering. If the `tabChildren` property is `undefined` or `true`, the children of a movie clip are included in automatic tab ordering. If the value of `tabChildren` is `false`, the children of a movie clip are not included in automatic tab ordering. The default value is `undefined`.

Availability

Flash Lite 2.0

Example

A list box UI widget built as a movie clip contains several items. The user can click each item to select it, so each item is a button. However, only the list box itself should be a tab stop. The items inside the list box should be excluded from tab ordering. To do this, the `tabChildren` property of the list box should be set to `false`.

The `tabChildren` property has no effect if the `tabIndex` property is used; the `tabChildren` property affects only automatic tab ordering.

The following example disables tabbing for all children movie clips inside a parent movie clip called `menu_mc`:

```
menu_mc.onRelease = function(){};
menu_mc.menu1_mc.onRelease = function(){};
menu_mc.menu2_mc.onRelease = function(){};
menu_mc.menu3_mc.onRelease = function(){};
menu_mc.menu4_mc.onRelease = function(){};

menu_mc.tabChildren = false;
```

Change the last line of code to the following to include the children movie clip instances of `menu_mc` in the automatic tab ordering:

```
menu_mc.tabChildren = true;
```

See also

tabIndex (Button.tabIndex property), tabEnabled (MovieClip.tabEnabled property), tabIndex (MovieClip.tabIndex property), tabIndex (TextField.tabIndex property)

tabEnabled (MovieClip.tabEnabled property)

```
public tabEnabled : Boolean
```

Specifies whether the movie clip is included in automatic tab ordering. It is `undefined` by default.

If the `tabEnabled` property is `undefined`, the object is included in automatic tab ordering only if it defines at least one movie clip handler, such as `MovieClip.onRelease`. If `tabEnabled` is `true`, the object is included in automatic tab ordering. If the `tabIndex` property is also set to a value, the object is included in custom tab ordering as well.

If `tabEnabled` is `false`, the object is not included in automatic or custom tab ordering, even if the `tabIndex` property is set. However, if `MovieClip.tabChildren` is `true`, the movie clip's children can still be included in automatic tab ordering, even if `tabEnabled` is `false`.

Availability
Flash Lite 2.0

Example
The following example does not include `myMC2_mc` in the automatic tab ordering:

```
myMC1_mc.onRelease = function() {};
myMC2_mc.onRelease = function() {};
myMC3_mc.onRelease = function() {};
myMC2_mc.tabEnabled = false;
```

See also
onRelease (MovieClip.onRelease handler), tabEnabled (Button.tabEnabled property), tabChildren (MovieClip.tabChildren property), tabIndex (MovieClip.tabIndex property), tabEnabled (TextField.tabEnabled property)

tabIndex (MovieClip.tabIndex property)

```
public tabIndex : Number
```

Lets you customize the tab ordering of objects in a movie. The `tabIndex` property is `undefined` by default. You can set the `tabIndex` property on a button, movie clip, or text field instance.

If an object in a SWF file contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the SWF file. The custom tab ordering includes only objects that have `tabIndex` properties.

The `tabIndex` property must be a positive integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` value of `1` precedes an object with a `tabIndex` value of `2`. The custom tab ordering disregards the hierarchical relationships of objects in a SWF file. All objects in the SWF file with `tabIndex` properties are placed in the tab order. Do not use the same `tabIndex` value for multiple objects.

Availability
Flash Lite 2.0

Example
The following ActionScript sets a custom tab order for three movie clip instances.

```
myMC1_mc.onRelease = function() {};
myMC2_mc.onRelease = function() {};
myMC3_mc.onRelease = function() {};
myMC1_mc.tabIndex = 2;
myMC2_mc.tabIndex = 1;
myMC3_mc.tabIndex = 3;
```

See also

tabIndex (Button.tabIndex property), tabIndex (TextField.tabIndex property)

_target (MovieClip._target property)

```
public _target : String [read-only]
```

Returns the target path of the movie clip instance, in slash notation. Use the `eval()` function to convert the target path to dot notation.

Availability

Flash Lite 2.0

Example

The following example displays the target paths of movie clip instances in a SWF file, in both slash and dot notation.

```
for (var i in this) {
    if (typeof (this[i]) == "movieclip") {
    trace("name: " + this[i]._name + ",\t target: " + this[i]._target + ",\t target(2):"
            + eval(this[i]._target));
    }
}
```

_totalframes (MovieClip._totalframes property)

```
public _totalframes : Number [read-only]
```

Returns the total number of frames in the movie clip instance specified in the `MovieClip` parameter.

Example

In the following example, two movie clip buttons control the Timeline. The `prev_mc` button moves the playhead to the previous frame, and the `next_mc` button moves the playhead to the next frame. Add content to a series of frames on the Timeline, and add the following ActionScript to Frame 1 of the Timeline:

```
stop();
prev_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe>1) {
    parent_mc.prevFrame();
    } else {
    parent_mc.gotoAndStop(parent_mc._totalframes);
    }
};
next_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe<parent_mc._totalframes) {
    parent_mc.nextFrame();
    } else {
    parent_mc.gotoAndStop(1);
    }
};
```

trackAsMenu (MovieClip.trackAsMenu property)

```
public trackAsMenu : Boolean
```

A Boolean value that indicates whether other buttons or movie clips can receive a release event from a mouse or stylus. If you drag a stylus or mouse across a movie clip and then release it on a second movie clip, the `onRelease` event is registered for the second movie clip. This allows you to create menus for the second movie clip. You can set the `trackAsMenu` property on any button or movie clip object. If you have not defined the `trackAsMenu` property, the default behavior is `false`.

You can change the `trackAsMenu` property at any time; the modified movie clip immediately takes on the new behavior.

Note: This property is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example sets the `trackAsMenu` property for three movie clips on the Stage. Click a movie clip and release the mouse button on a second movie clip to see which instance receives the event.

```
myMC1_mc.trackAsMenu = true;
myMC2_mc.trackAsMenu = true;
myMC3_mc.trackAsMenu = false;

myMC1_mc.onRelease = clickMC;
myMC2_mc.onRelease = clickMC;
myMC3_mc.onRelease = clickMC;

function clickMC() {
    trace("you clicked the "+this._name+" movie clip.");
};
```

See also

`trackAsMenu (Button.trackAsMenu property)`

transform (MovieClip.transform property)

```
public transform : Transform
```

An object with properties pertaining to a movie clip's matrix, color transform, and pixel bounds. The specific properties matrix, colorTransform, and three read-only properties (`concatenatedMatrix`, `concatenatedColorTransform`, and `pixelBounds`) are described in the entry for the Transform class.

Each of the transform object's properties is itself an object. This is important because the only way to set new values for the matrix or colorTransform objects is to create an object and copy that object into the transform.matrix or transform.colorTransform property.

For example, to increase the `tx` value of a movie clip's matrix, you must make a copy of the entire matrix object, modify the `tx` property of the new object, and then copy the new object into the matrix property of the transform object:

```
var myMatrix:Object = myDisplayObject.transform.matrix;
myMatrix.tx += 10;
myDisplayObject.transform.matrix = myMatrix;
```

You cannot directly set the `tx` property. The following code has no effect on `myDisplayObject`:

```
myDisplayObject.transform.matrix.tx += 10;
```

You can also copy an entire transform object and assign it to another movie clip's transform property. For example, the following code copies the entire transform object from `myOldDisplayObj` to `myNewDisplayObj`:

```
myNewDisplayObj.transform = myOldDisplayObj.transform;
```

The new movie clip, `myNewDisplayObj`, now has the same values for its matrix, color transform, and pixel bounds as the old movie clip, `myOldDisplayObj`.

Availability
Flash Lite 3.1

Example
The following example shows how to use a movie clip's `transform` property to access and modify a movie clip's location by using Matrix positioning.

```
import flash.geom.Matrix;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var translateMatrix:Matrix = new Matrix();
translateMatrix.translate(10, 0);

rect.onPress = function() {
    var tmpMatrix:Matrix = this.transform.matrix;
    tmpMatrix.concat(translateMatrix);
    this.transform.matrix = tmpMatrix;
}

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

See also
"Transform (flash.geom.Transform)" on page 652

unloadMovie (MovieClip.unloadMovie method)

```
public unloadMovie() : Void
```

Removes the contents of a movie clip instance. The instance properties and clip handlers remain.

To remove the instance, including its properties and clip handlers, use `MovieClip.removeMovieClip()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass.

Availability
Flash Lite 2.0

Example

The following example unloads a movie clip instance called `box` when a user clicks the `box` movie clip:

```
this.createEmptyMovieClip("box", 1);

with (box) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0x4827CF);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
}

box.onRelease = function() {
        box.unloadMovie();
};
```

See also

removeMovieClip (MovieClip.removeMovieClip method), attachMovie (MovieClip.attachMovie method), loadMovie (MovieClip.loadMovie method), unloadMovie function, unloadMovieNum function

_url (MovieClip._url property)

```
public _url : String [read-only]
```

Retrieves the URL of the SWF, JPEG, GIF, or PNG file from which the movie clip was downloaded.

Availability

Flash Lite 2.0

Example

The following example displays the URL of the image that is loaded into the `image_mc` instance in the Output panel.

```
this.createEmptyMovieClip("image_mc", 1);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace("_url: "+target_mc._url);
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg", image_mc);
```

_visible (MovieClip._visible property)

```
public _visible : Boolean
```

A Boolean value that indicates whether the movie clip is visible. Movie clips that are not visible (`_visible` property set to `false`) are disabled. For example, a button in a movie clip with `_visible` set to `false` cannot be clicked.

Availability

Flash Lite 2.0

Example

The following example sets the `_visible` property for two movie clips called `myMC1_mc` and `myMC2_mc`. The property is set to `true` for one instance, and `false` for the other. Notice that `myMC1_mc` instance cannot be clicked after the `_visible` property is set to `false`.

```
myMC1_mc.onRelease = function() {
    trace(this._name+"._visible = false");
    this._visible = false;
};
myMC2_mc.onRelease = function() {
    trace(this._name+"._alpha = 0");
    this._alpha = 0;
};
```

See also

`_visible (Button._visible property)`, `_visible (TextField._visible property)`

_width (MovieClip._width property)

```
public _width : Number
```

The width of the movie clip, in pixels.

Availability

Flash Lite 2.0

Example

The following code example displays the height and width of a movie clip in the Output panel:

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());

triangle.beginFill(0x0000FF, 100);
triangle.moveTo(100, 100);
triangle.lineTo(100, 150);
triangle.lineTo(150, 100);
triangle.lineTo(100, 100);

trace(triangle._name + " = " + triangle._width + " X " + triangle._height + " pixels");
```

See also

`_height (MovieClip._height property)`

_x (MovieClip._x property)

```
public _x : Number
```

An integer that sets the *x* coordinate of a movie clip relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, its coordinate system refers to the upper-left corner of the Stage as (0, 0). If the move clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

Availability
Flash Lite 2.0

See also
_xscale (MovieClip._xscale property), _y (MovieClip._y property), _yscale (MovieClip._yscale property)

_xmouse (MovieClip._xmouse property)

```
public _xmouse : Number [read-only]
```

Returns the *x* coordinate of the mouse position.

Note: This property is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
The following example returns the current *x* and *y* coordinates of the mouse on the Stage (`_level0`) and in relation to a movie clip on the Stage called `my_mc`:

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 0, 0, 150, 66);
mouse_txt.html = true;
mouse_txt.multiline = true;
var row1_str:String = " \t<b>_xmouse\t</b><b>_ymouse</b>";
my_mc.onMouseMove = function() {
    mouse_txt.htmlText = "<textformat tabStops='[50,100]'>";
    mouse_txt.htmlText += row1_str;
    mouse_txt.htmlText += "<b>_level0</b>\t"+_xmouse+"\t"+_ymouse;
    mouse_txt.htmlText += "<b>my_mc</b>\t"+this._xmouse+"\t"+this._ymouse;
    mouse_txt.htmlText += "</textformat>";
};
```

See also
hasMouse (capabilities.hasMouse property), _ymouse (MovieClip._ymouse property)

_xscale (MovieClip._xscale property)

```
public _xscale : Number
```

Sets the horizontal scale (`percentage`) of the movie clip as applied from the registration point of the movie clip. The default registration point is (0,0).

Scaling the local coordinate system affects the `_x` and `_y` property settings, which are defined in whole pixels. For example, if the parent movie clip is scaled to 50%, setting the `_y` property moves an object in the movie clip by half the number of pixels that it would if the movie were set at 100%.

Availability
Flash Lite 2.0

Example

The following example creates a movie clip called `box_mc` at runtime. The Drawing API is used to draw a box in this instance, and when the mouse rolls over the box, horizontal and vertical scaling is applied to the movie clip. When the mouse rolls off the instance, it returns to the previous scaling.

```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0xEEEEEE);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
};
box_mc.onRollOver = function() {
    this._x -= this._width/2;
    this._y -= this._height/2;
    this._xscale = 200;
    this._yscale = 200;
};
box_mc.onRollOut = function() {
    this._xscale = 100;
    this._yscale = 100;
    this._x += this._width/2;
    this._y += this._height/2;
};
```

See also

_x (MovieClip._x property), _y (MovieClip._y property), _yscale (MovieClip._yscale property), _width (MovieClip._width property)

_y (MovieClip._y property)

```
public _y : Number
```

Sets the *y* coordinate of a movie clip relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, its coordinate system refers to the upper-left corner of the Stage.as (0,0). If the movie clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

Availability

Flash Lite 2.0

See also

_x (MovieClip._x property), _xscale (MovieClip._xscale property), _yscale (MovieClip._yscale property)

_ymouse (MovieClip._ymouse property)

```
public _ymouse : Number [read-only]
```

Indicates the *y* coordinate of the mouse position.

Note: This property is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example returns the current *x* and *y* coordinates of the mouse on the Stage (`_level0`) and in relation to a movie clip on the Stage called `my_mc`.

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 0, 0, 150, 66);
mouse_txt.html = true;
mouse_txt.multiline = true;
var row1_str:String = " \t<b>_xmouse\t</b><b>_ymouse</b>";
my_mc.onMouseMove = function() {
    mouse_txt.htmlText = "<textformat tabStops='[50,100]'>";
    mouse_txt.htmlText += row1_str;
    mouse_txt.htmlText += "<b>_level0</b>\t"+_xmouse+"\t"+_ymouse;
    mouse_txt.htmlText += "<b>my_mc</b>\t"+this._xmouse+"\t"+this._ymouse;
    mouse_txt.htmlText += "</textformat>";
};
```

See also

hasMouse (capabilities.hasMouse property), _xmouse (MovieClip._xmouse property)

_yscale (MovieClip._yscale property)

```
public _yscale : Number
```

Sets the vertical scale (`percentage`) of the movie clip as applied from the registration point of the movie clip. The default registration point is (0,0).

Scaling the local coordinate system affects the `_x` and `_y` property settings, which are defined in whole pixels. For example, if the parent movie clip is scaled to 50%, you set the `_x` property to move an object in the movie clip by half the number of pixels that it would if the movie were at 100%.

Availability

Flash Lite 2.0

Example

The following example creates a movie clip at runtime called `box_mc`. The Drawing API is used to draw a box in this instance, and when the mouse rolls over the box, horizontal and vertical scaling is applied to the movie clip. When the mouse rolls off the instance, it returns to the previous scaling.

```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0xEEEEEE);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
};
box_mc.onRollOver = function() {
    this._x -= this._width/2;
    this._y -= this._height/2;
    this._xscale = 200;
    this._yscale = 200;
};
box_mc.onRollOut = function() {
    this._xscale = 100;
    this._yscale = 100;
    this._x += this._width/2;
    this._y += this._height/2;
};
```

See also

_x (MovieClip._x property), _xscale (MovieClip._xscale property), _y (MovieClip._y property), _height (MovieClip._height property)

MovieClipLoader

```
Object
    |
    +-MovieClipLoader
```

```
public class MovieClipLoader
extends Object
```

The MovieClipLoader class lets you implement listener callbacks that provide status information while SWF, JPEG, GIF, and PNG files are being loaded (downloaded) into movie clips. To use MovieClipLoader features, use `MovieClipLoader.loadClip()` instead of `loadMovie()` or `MovieClip.loadMovie()` to load SWF files.

After you issue the `MovieClipLoader.loadClip()` method, the following events take place in the order listed:

- When the first bytes of the downloaded file are written to disk, the `MovieClipLoader.onLoadStart` listener is invoked.
- If you implemented the `MovieClipLoader.onLoadProgress` listener, it is invoked during the loading process.
- **Note:** You can call `MovieClipLoader.getProgress()` at any time during the load process.
- When the entire downloaded file is written to disk, the `MovieClipLoader.onLoadComplete` listener is invoked.
- After the downloaded file's first frame actions are executed, the `MovieClipLoader.onLoadInit` listener is invoked.

After `MovieClipLoader.onLoadInit` is invoked, you can set properties, use methods, and otherwise interact with the loaded movie.

If the file fails to load completely, the `MovieClipLoader.onLoadError` listener is invoked.

Availability

Flash Lite 2.0

Property summary

Properties inherited from class Object

```
constructor (Object.constructor property), __proto__ (Object.__proto__ property),
prototype (Object.prototype property), __resolve (Object.__resolve property)
```

Event summary

Event	Description
onLoadComplete = function(listenerObject, [target_mc]) {}	Invoked when a file loaded with `MovieClipLoader.loadClip()` is completely downloaded.
onLoadError = function(target_mc, errorCode) {}	Invoked when a file loaded with `MovieClipLoader.loadClip()` has failed to load.
onLoadInit = function([target_mc]) {}	Invoked when the actions on the first frame of the loaded clip are executed.
onLoadProgress = function([target_mc], loadedBytes, totalBytes) {}	Invoked every time the loading content is written to disk during the loading process (that is, between `MovieClipLoader.onLoadStart` and `MovieClipLoader.onLoadComplete`).
onLoadStart = function([target_mc]) {}	Invoked when a call to `MovieClipLoader.loadClip()` has successfully begun to download a file.

Constructor summary

Signature	Description
MovieClipLoader()	Creates a MovieClipLoader object that you can use to implement a number of listeners to respond to events while a SWF, JPEG, GIF, or PNG file is downloading.

Method summary

Modifiers	Signature	Description
	addListener(listener:Object) : Boolean	Registers an object to receive notification when a MovieClipLoader event handler is invoked.
	getProgress(target:Object) : Object	Returns the number of bytes loaded and total number of bytes for a file that is being loaded by using MovieClipLoader.loadClip(); for compressed movies, the getProgress method reflects the number of compressed bytes.
	loadClip(url:String, target:Object) : Boolean	Loads a SWF or JPEG file into a movie clip in Flash Lite player while the original movie is playing.
	removeListener(listener:Object) : Boolean	Removes the listener that was used to receive notification when a MovieClipLoader event handler was invoked.
	unloadClip(target:Object) : Boolean	Removes a movie clip that was loaded by means of MovieClipLoader.loadClip().

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

addListener (MovieClipLoader.addListener method)

```
public addListener(listener:Object) : Boolean
```

Registers an object to receive notification when a MovieClipLoader event handler is invoked.

Availability

Flash Lite 2.0

Parameters

listener : Object - An object that listens for a callback notification from the MovieClipLoader event handlers.

Returns

Boolean - A Boolean value. The return value is `true` if the listener was established successfully; otherwise the return value is `false`.

Example

The following example loads an image into a movie clip called `image_mc`. The movie clip instance is rotated and centered on the Stage, and both the Stage and movie clip have a stroke drawn around their perimeters.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._x = Stage.width/2-target_mc._width/2;
    target_mc._y = Stage.height/2-target_mc._width/2;
    var w:Number = target_mc._width;
    var h:Number = target_mc._height;
    target_mc.lineStyle(4, 0x000000);
    target_mc.moveTo(0, 0);
    target_mc.lineTo(w, 0);
    target_mc.lineTo(w, h);
    target_mc.lineTo(0, h);
    target_mc.lineTo(0, 0);
    target_mc._rotation = 3;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg", image_mc);
```

See also

onLoadComplete (MovieClipLoader.onLoadComplete event listener), onLoadError (MovieClipLoader.onLoadError event listener), onLoadInit (MovieClipLoader.onLoadInit event listener), onLoadProgress (MovieClipLoader.onLoadProgress event listener), onLoadStart (MovieClipLoader.onLoadStart event listener), removeListener (MovieClipLoader.removeListener method)

getProgress (MovieClipLoader.getProgress method)

```
public getProgress(target:Object) : Object
```

Returns the number of bytes loaded and total number of bytes for a file that is being loaded by using `MovieClipLoader.loadClip()`; for compressed movies, the `getProgress` method reflects the number of compressed bytes. The `getProgress` method lets you explicitly request this information, instead of (or in addition to) writing a `MovieClipLoader.onLoadProgress` listener function.

Availability

Flash Lite 2.0

Parameters

target : Object - A SWF, JPEG, GIF, or PNG file that is loaded using `MovieClipLoader.loadClip()`.

Returns

Object - An object that has two integer properties: `bytesLoaded` and `bytesTotal`.

Example

The following example demonstrates usage of the `getProgress` method. Rather than using this method, one will usually create a listener object and listen for the `onLoadProgress` event. Another important note about this method, is that the first, synchronous call to getProgress can return the bytesLoaded and bytesTotal of the *container* and not the values for the externally requested object.

```
var container:MovieClip = this.createEmptyMovieClip("container", this.getNextHighestDepth());
var image:MovieClip = container.createEmptyMovieClip("image", container.getNextHighestDepth());

var mcLoader:MovieClipLoader = new MovieClipLoader();
var listener:Object = new Object();
listener.onLoadProgress = function(target:MovieClip, bytesLoaded:Number,
bytesTotal:Number):Void {
    trace(target + ".onLoadProgress with " + bytesLoaded + " bytes of " + bytesTotal);
}
mcLoader.addListener(listener);
mcLoader.loadClip("http://www.w3.org/Icons/w3c_main.png", image);

var interval:Object = new Object();
interval.id = setInterval(checkProgress, 100, mcLoader, image, interval);

function checkProgress(mcLoader:MovieClipLoader, image:MovieClip, interval:Object):Void {
    trace(">> checking progress now with : " + interval.id);
    var progress:Object = mcLoader.getProgress(image);
    trace("bytesLoaded: " + progress.bytesLoaded + " bytesTotal: " + progress.bytesTotal);
    if(progress.bytesLoaded == progress.bytesTotal) {
        clearInterval(interval.id);
    }
}
```

See also

`loadClip (MovieClipLoader.loadClip method)`, `onLoadProgress (MovieClipLoader.onLoadProgress event listener)`

loadClip (MovieClipLoader.loadClip method)

```
public loadClip(url:String, target:Object) : Boolean
```

Loads a SWF or JPEG file into a movie clip in Flash Lite player while the original movie is playing. Using this method lets you display several SWF files at once and switch between SWF files without loading another HTML document.

Using the `loadClip()` method instead of `loadMovie()` or `MovieClip.loadMovie()` has a number of advantages. The following handlers are implemented by the use of a listener object. You activate the listener by using `MovieClipLoader.addListener(listenerObject)` to register it with the MovieClipLoader class.

- The `MovieClipLoader.onLoadStart` handler is invoked when loading begins.
- The `MovieClipLoader.onLoadError` handler is invoked if the clip cannot be loaded.
- The `MovieClipLoader.onLoadProgress` handler is invoked as the loading process progresses.
- The `MovieClipLoader.onLoadComplete` handler is invoked when a file completes downloading, but before the loaded movie clip's methods and properties are available. This handler is called before the `onLoadInit` handler.
- The `MovieClipLoader.onLoadInit` handler is invoked after the actions in the first frame of the clip are executed, so you can begin manipulating the loaded clip. This handler is called after the `onLoadComplete` handler. For most purposes, use the `onLoadInit` handler.

A SWF file or image loaded into a movie clip inherits the position, rotation, and scale properties of the movie clip. You can use the target path of the movie clip to target the loaded movie.

You can use the `loadClip()` method to load one or more files into a single movie clip or level; MovieClipLoader listener objects are passed to the loading target movie clip instance as a parameter. Alternatively, you can create a different MovieClipLoader object for each file that you load.

Use `MovieClipLoader.unloadClip()` to remove movies or images loaded with this method or to cancel a load operation that is in progress.

`MovieClipLoader.getProgress()` and `MovieClipLoaderListener.onLoadProgress` do not report the actual `bytesLoaded` and `bytesTotal` values in the Authoring player when the files are local. When you use the Bandwidth Profiler feature in the authoring environment, `MovieClipLoader.getProgress()` and `MovieClipLoaderListener.onLoadProgress` report the download at the actual download rate, not at the reduced bandwidth rate that the Bandwidth Profiler provides.

Availability
Flash Lite 2.0

Parameters
url : String - The absolute or relative URL of the SWF or JPEG file to be loaded. A relative path must be relative to the SWF file at level 0. Absolute URLs must include the protocol reference, such as http:// or file:///. Filenames cannot include disk drive specifications.

target : Object - The target path of a movie clip, or an integer specifying the level in Flash Lite player into which the movie will be loaded. The target movie clip is replaced by the loaded SWF file or image.

Returns
Boolean - A Boolean value. The return value is `true` if the URL request was sent successfully; otherwise the return value is `false`.

Example
The following example shows you how to use the `MovieClipLoader.loadClip` method by creating handler for the `onLoadInit` event and then making the request.

The following code should either be placed directly into a frame action on a timeline, or pasted into a class that extends MovieClip.

Create a handler method for the `onLoadInit` event.

```
public function onLoadInit(mc:MovieClip):Void {
    trace("onLoadInit: " + mc);
}
```

Create an empty MovieClip and use the MovieClipLoader to load an image into it.

```
var container:MovieClip = createEmptyMovieClip("container", getNextHighestDepth());
var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(this);
mcLoader.loadClip("YourImage.jpg", container);

function onLoadInit(mc:MovieClip) {
    trace("onLoadInit: " + mc);
}
```

See also
onLoadInit (MovieClipLoader.onLoadInit event listener)

MovieClipLoader constructor

```
public MovieClipLoader()
```

Creates a MovieClipLoader object that you can use to implement a number of listeners to respond to events while a SWF, JPEG, GIF, or PNG file is downloading.

Availability
Flash Lite 2.0

Example
See `MovieClipLoader.loadClip()`.

See also
`addListener (MovieClipLoader.addListener method)`, `loadClip (MovieClipLoader.loadClip method)`

onLoadComplete (MovieClipLoader.onLoadComplete event listener)

```
onLoadComplete = function(listenerObject, [target_mc]) {}
```

Invoked when a file loaded with `MovieClipLoader.loadClip()` is completely downloaded. The value for `target_mc` identifies the movie clip for which this call is being made. This is useful if multiple files are being loaded with the same set of listeners.

This parameter is passed by Flash to your code, but you do not have to implement all of the parameters in the listener function.

When you use the `onLoadComplete` and `onLoadInit` events with the MovieClipLoader class, it's important to understand how this differs from the way they work with your SWF file. The `onLoadComplete` event is called after the SWF or JPEG file is loaded, but before the application is initialized. At this point you cannot access the loaded movie clip's methods and properties, and because of this you cannot call a function, move to a specific frame, and so on. In most situations, it's better to use the `onLoadInit` event instead, which is called after the content is loaded and is fully initialized.

Availability
Flash Lite 2.0

Parameters
listenerObject: - A listener object that was added using `MovieClipLoader.addListener()`.

target_mc: [optional] - A movie clip loaded by a `MovieClipLoader.loadClip()` method. This parameter is optional.

Example
The following example loads an image into a movie clip instance called `image_mc`. The `onLoadInit` and `onLoadComplete` events are used to determine how long it takes to load the image. The information appears in a dynamically created text field called timer_txt.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
target_mc._height, target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/112x112/box_studio
_112x112.jpg", image_mc);
```

See also

addListener (MovieClipLoader.addListener method), loadClip (MovieClipLoader.loadClip method), onLoadStart (MovieClipLoader.onLoadStart event listener), onLoadError (MovieClipLoader.onLoadError event listener), onLoadInit (MovieClipLoader.onLoadInit event listener)

onLoadError (MovieClipLoader.onLoadError event listener)

```
onLoadError = function(target_mc, errorCode) {}
```

Invoked when a file loaded with `MovieClipLoader.loadClip()` has failed to load. This listener can be invoked for various reasons, including if the server is down, if the file is not found, or if a security violation occurs.

Call this listener on a listener object that you add using `MovieClipLoader.addListener()`.

The value for `target_mc` identifies the movie clip this call is being made for. This parameter is useful if you are loading multiple files with the same set of listeners.

For the `errorCode` parameter, the string "URLNotFound" is returned if neither the `MovieClipLoader.onLoadStart` or `MovieClipLoader.onLoadComplete` listener is called, for example, if a server is down or the file is not found. The string `"LoadNeverCompleted"` is returned if `MovieClipLoader.onLoadStart` was called but `MovieClipLoader.onLoadComplete` was not called, for example, if the download was interrupted because of server overload, server crash, and so on.

Availability

Flash Lite 2.0

Parameters

target_mc: - A movie clip loaded by a `MovieClipLoader.loadClip()` method.

errorCode: - A string that explains the reason for the failure.

Example

The following example displays information in the Output panel when an image fails to load.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
    trace("ERROR!");
    switch (errorCode) {
    case 'URLNotFound' :
        trace("\t Unable to connect to URL: "+target_mc._url);
        break;
    case 'LoadNeverCompleted' :
        trace("\t Unable to complete download: "+target_mc);
        break;
    }
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace("success");
    trace(image_mcl.getProgress(target_mc).bytesTotal+" bytes loaded");
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.fakedomain.com/images/bad_hair_day.jpg", image_mc);
```

See also

addListener (MovieClipLoader.addListener method), loadClip (MovieClipLoader.loadClip method), onLoadStart (MovieClipLoader.onLoadStart event listener), onLoadComplete (MovieClipLoader.onLoadComplete event listener)

onLoadInit (MovieClipLoader.onLoadInit event listener)

```
onLoadInit = function([target_mc]) {}
```

Invoked when the actions on the first frame of the loaded clip are executed. After this listener is invoked, you can set properties, use methods, and otherwise interact with the loaded movie. Call this listener on a listener object that you add using `MovieClipLoader.addListener()`.

The value for `target_mc` identifies the movie clip this call is being made for. This parameter is useful if you are loading multiple files with the same set of listeners.

Availability

Flash Lite 2.0

Parameters

target_mc: [optional] - A movie clip loaded by a `MovieClipLoader.loadClip()` method.

Example

The following example loads an image into a movie clip instance called `image_mc`. The `onLoadInit` and `onLoadComplete` events are used to determine how long it takes to load the image. This information appears in a text field called `timer_txt`.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
target_mc._height,
target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg", image_mc);
```

The following example checks whether a movie is loaded into a movie clip created at runtime:

```
this.createEmptyMovieClip("tester_mc", 1);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace("movie loaded");
}
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.yourserver.com/your_movie.swf", tester_mc);
```

See also

addListener (MovieClipLoader.addListener method), loadClip (MovieClipLoader.loadClip method), onLoadStart (MovieClipLoader.onLoadStart event listener)

onLoadProgress (MovieClipLoader.onLoadProgress event listener)

```
onLoadProgress = function([target_mc], loadedBytes, totalBytes) {}
```

Invoked every time the loading content is written to disk during the loading process (that is, between `MovieClipLoader.onLoadStart` and `MovieClipLoader.onLoadComplete`). Call this listener on a listener object that you add using `MovieClipLoader.addListener()`. You can use this method to display information about the progress of the download, using the `loadedBytes` and `totalBytes` parameters.

The value for `target_mc` identifies the movie clip this call is being made for. This is useful if you are loading multiple files with the same set of listeners.

Note: If you attempt to use `onLoadProgress` in test movie mode with a local file that resides on your hard disk, it will not work properly because, in test movie mode, Flash Lite player loads local files in their entirety.

Parameters

`target_mc: MovieClip [optional]` A movie clip loaded by a `MovieClipLoader.loadClip()` method.

`loadedBytes: Number` The number of bytes that had been loaded when the listener was invoked.

`totalBytes: Number` The total number of bytes in the file being loaded.

Availability
Flash Lite 2.0

Parameters
target_mc: [optional] - A movie clip loaded by a `MovieClipLoader.loadClip()` method.

loadedBytes: - The number of bytes that had been loaded when the listener was invoked.

totalBytes: - The total number of bytes in the file being loaded.

Example
The following example creates a new movie clip, a new `MovieClipLoader` and an anonymous event listener. It should periodically output the progress of a load and finally provide notification when the load is complete and the asset is available to ActionScript.

```
var container:MovieClip = this.createEmptyMovieClip("container", this.getNextHighestDepth());
var mcLoader:MovieClipLoader = new MovieClipLoader();
var listener:Object = new Object();
listener.onLoadProgress = function(target:MovieClip, bytesLoaded:Number,
bytesTotal:Number):Void {
    trace(target + ".onLoadProgress with " + bytesLoaded + " bytes of " + bytesTotal);
}
listener.onLoadInit = function(target:MovieClip):Void {
    trace(target + ".onLoadInit");
}
mcLoader.addListener(listener);
mcLoader.loadClip("http://www.w3.org/Icons/w3c_main.png", container);
```

See also
`addListener (MovieClipLoader.addListener method)`, `loadClip (MovieClipLoader.loadClip method)`, `getProgress (MovieClipLoader.getProgress method)`

onLoadStart (MovieClipLoader.onLoadStart event listener)

```
onLoadStart = function([target_mc]) {}
```

Invoked when a call to `MovieClipLoader.loadClip()` has successfully begun to download a file. Call this listener on a listener object that you add using `MovieClipLoader.addListener()`.

The value for `target_mc` identifies the movie clip this call is being made for. This parameter is useful if you are loading multiple files with the same set of listeners.

Availability
Flash Lite 2.0

Parameters
target_mc: [optional] - A movie clip loaded by a `MovieClipLoader.loadClip()` method.

Example
The following example loads an image into a movie clip instance called `image_mc`. The `onLoadInit` and `onLoadComplete` events are used to determine how long it takes to load the image. This information appears in a text field called `timer_txt`.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
target_mc._height,
target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg", image_mc);
```

See also

addListener (MovieClipLoader.addListener method), loadClip (MovieClipLoader.loadClip method), onLoadError (MovieClipLoader.onLoadError event listener), onLoadInit (MovieClipLoader.onLoadInit event listener) onLoadComplete (MovieClipLoader.onLoadComplete event listener)

removeListener (MovieClipLoader.removeListener method)

```
public removeListener(listener:Object) : Boolean
```

Removes the listener that was used to receive notification when a `MovieClipLoader` event handler was invoked. No further loading messages will be received.

Availability

Flash Lite 2.0

Parameters

listener : Object - A listener object that was added using `MovieClipLoader.addListener()`.

Returns

Boolean -

Example

The following example loads an image into a movie clip, and enables the user to start and stop the loading process using two buttons called `start_button` and `stop_button`. When the user starts or stops the progress, information is displayed in the Output panel.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
    trace("\t onLoadStart");
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    trace("\t onLoadComplete");
};
mclListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
    trace("\t onLoadError: "+errorCode);
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace("\t onLoadInit");
    start_button.enabled = true;
    stop_button.enabled = false;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
//
start_button.clickHandler = function() {
    trace("Starting...");
    start_button.enabled = false;
    stop_button.enabled = true;
    //
    image_mcl.addListener(mclListener);
    image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg", image_mc);
};
stop_button.clickHandler = function() {
    trace("Stopping...");
    start_button.enabled = true;
    stop_button.enabled = false;
    //
    image_mcl.removeListener(mclListener);
};
stop_button.enabled = false;
```

See also

addListener (MovieClipLoader.addListener method)

unloadClip (MovieClipLoader.unloadClip method)

```
public unloadClip(target:Object) : Boolean
```

Removes a movie clip that was loaded by means of `MovieClipLoader.loadClip()`. If you call this method while a movie is loading, `MovieClipLoader.onLoadError` is invoked.

Availability

Flash Lite 2.0

Parameters

target : Object - The string or integer passed to the corresponding call to `my_mcl.loadClip()`.

Returns

Boolean -

Example

The following example loads an image into a movie clip called `image_mc`. If you click the movie clip, the movie clip is removed and information is displayed in the Output panel.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._x = 100;
    target_mc._y = 100;
    target_mc.onRelease = function() {
    trace("Unloading clip...");
    trace("\t name: "+target_mc._name);
    trace("\t url: "+target_mc._url);
    image_mcl.unloadClip(target_mc);
    };
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg", image_mc);
```

See also

loadClip (MovieClipLoader.loadClip method), onLoadError (MovieClipLoader.onLoadError event listener)

NetConnection

Creates a NetConnection object that you can use with a NetStream object to invoke commands on a remote application server or to play back streaming video (FLV) files either locally or from a server.

Method	Description
`connect();`	`connect(command:String, ... arguments):void` Opens a connection to a server.
`close();`	`close():void` Closes the connection that was opened locally or with the server and dispatches the netStatus event with a code property of `NetConnection.Connect.Closed`.

Availability

Flash Lite 3.0

connect (NetConnection.connect) method

Opens a connection to a server. Through this connection, you can play back audio or video (FLV) files from the local file system, or you can invoke commands on a remote server.

When using this method, consider the Flash Lite player security model and the following security considerations:

- By default, the website denies access between sandboxes. The website can enable access to a resource by using a cross-domain policy file.
- A website can deny access to a resource by adding server-side ActionScript application logic in Flash Media Server.

- You cannot use the `NetConnection.connect()` method if the calling SWF file is in the local-with-file-system sandbox.
- You can prevent a SWF file from using this method by setting the `allowNetworking` parameter of the object and embed tags in the HTML page that contains the SWF content.

Availability

Flash Lite 3.0

close (NetConnection.close method)

```
public function close():void
```

Closes the connection that was opened locally or with the server and dispatches the `netStatus` event with a code property of `NetConnection.Connect.Closed`.

This method disconnects all NetStream objects running over this connection; any queued data that has not been sent is discarded. (To terminate streams without closing the connection, use `NetStream.close()`.) If you call this method and then want to reconnect, you must recreate the NetStream object.

Availability

Flash Lite 3.0

See also

NetStream

NetConnection constructor

```
public "NetConnection" on page 472()
```

Creates a NetConnection object that you can use in conjunction with a NetStream object to play back local streaming video (FLV) files. After creating the NetConnection object, use the `Connect(NetConnection.connect)` method to make the actual connection.

Playing external FLV files provides several advantages over embedding video in a Flash document, such as better performance and memory management, and independent video and Flash frame rates. The NetConnection class provides the means to play back streaming FLV files from a local drive or HTTP address.

Availability

Flash Lite 3.0

Example

See the example for `connect (NetConnection.connect)` method.

See also

"connect (NetConnection.connect) method" on page 472, "attachVideo (Video.attachVideo method)" on page 661, "NetStream" on page 474

NetStream

Creates a stream that can be used for playing FLV files through the specified NetConnection object.

Local file URLs are also supported by simply replacing "http:" with "file:" For example:

```
NetStream.play("http://somefile.flv");
NetStream.play("file://somefile.flv");
```

Note: *Standard security restrictions apply. For example a remote SWF file cannot access absolute file:// URLs in the form of "file://C:/somefile.flv".*

Availability

Flash Lite 3.0

NetStream Class methods

Method	Description
`close()`	`close():void` Stops playing all data on the stream, sets the time property to 0, and makes the stream available for another use.
`pause()`	`pause():void` Pauses playback of a video stream.
`play()`	`play(... arguments):void` Begins playback of external audio or a video (FLV) file.
`seek()`	Seeks the keyframe closest to the specified number of seconds from the beginning of the stream.
`setBufferTime()`	Specifies how long to buffer messages before starting to display.

NetStream Class properties

Properties	Description
`bufferLength`	The number of seconds of data currently in the buffer.
`bufferTime`	The number of seconds assigned to the buffer by `setBufferTime()`.
`bytesLoaded`	The number of bytes of data that have been loaded into the player.
`bytesTotal`	The total size in bytes of the file being loaded into the player.
`currentFPS`	The number of frames per second being displayed.
`time`	The position of the playhead, in seconds.

NetStream Class events

Event	Description
`onStatus`	Invoked when a status change or error is posted for the NetStream object.
`onCuePoint`	Invoked when an embedded cue point is reached during the playing of an FLV.
`OnMetaData`	Invoked when Flash Lite player receives descriptive information embedded in the FLV.

bufferLength (NetStream.bufferLength property)

```
public bufferLength : Number [read-only]
```

The number of seconds of data currently in the buffer. You can use this property in conjunction with `NetStream.bufferTime` to estimate how close the buffer is to being full—for example, to display feedback to a user who is waiting for data to be loaded into the buffer.

Availability

Flash Lite 3.0

Note: *This property is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Example

The following example dynamically creates a text field that displays information about the number of seconds that are currently in the buffer. The text field also displays the buffer length that the video is set to, and percentage of buffer that is filled.

```
this.createTextField("buffer_txt", this.getNextHighestDepth(), 10, 10, 300, 22);
buffer_txt.html = true;

var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
stream_ns.setBufferTime(3);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

var buffer_interval:Number = setInterval(checkBufferTime, 100, stream_ns);
function checkBufferTime(my_ns:NetStream):Void {
      var bufferPct:Number = Math.min(Math.round(my_ns.bufferLength/my_ns.bufferTime*100), 100);
       var output_str:String = "<textformat tabStops='[100,200]'>";
       output_str += "Length: "+my_ns.bufferLength+"\t"+"Time:
"+my_ns.bufferTime+"\t"+"Buffer:"+bufferPct+"%";
       output_str += "</textformat>";
       buffer_txt.htmlText = output_str;
}
```

bufferTime (NetStream.bufferTime property)

```
public bufferTime : Number [read-only]
```

The number of seconds assigned to the buffer by `NetStream.setBufferTime()`. The default value is .1 (one-tenth of a second). To determine the number of seconds currently in the buffer, use `NetStream.bufferLength`.

Availability

Flash Lite 3.0

Note: *This property is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Example

The following example dynamically creates a text field that displays information about the number of seconds that are currently in the buffer. The text field also displays the buffer length that the video is set to, and percentage of buffer that is filled.

```
this.createTextField("buffer_txt", this.getNextHighestDepth(), 10, 10, 300, 22);
buffer_txt.html = true;

var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
stream_ns.setBufferTime(3);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

var buffer_interval:Number = setInterval(checkBufferTime, 100, stream_ns);
function checkBufferTime(my_ns:NetStream):Void {
var bufferPct:Number = Math.min(Math.round(my_ns.bufferLength
my_ns.bufferTime*100), 100);
var output_str:String = "<textformat tabStops='[100,200]'>";
output_str += "Length: "+my_ns.bufferLength+"\t"+"Time:
"+my_ns.bufferTime+"\t"+"Buffer:"+bufferPct+"%";
output_str += "</textformat>";
buffer_txt.htmlText = output_str;
}
```

bytesLoaded (NetStream.bytesLoaded property)

```
public bytesLoaded : Number [read-only]
```

The number of bytes of data that have been loaded into the player. You can use this method in conjunction with `NetStream.bytesTotal` to estimate how close the buffer is to being full--for example, to display feedback to a user who is waiting for data to be loaded into the buffer.

Availability

Flash Lite 3.0

Example

The following example creates a progress bar using the Drawing API and the `bytesLoaded` and `bytesTotal` properties. The bar displays the progress of the operation as video1.flv is loaded into the video object instance called my_video. A text field called `loaded_txt` is dynamically created to display information about the loading progress as well.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
progressBar_mc.createEmptyMovieClip("bar_mc", progressBar_mc.getNextHighestDepth());
with (progressBar_mc.bar_mc) {
       beginFill(0xFF0000);
       moveTo(0, 0);
       lineTo(100, 0);
       lineTo(100, 10);
       lineTo(0, 10);
       lineTo(0, 0);
       endFill();
       _xscale = 0;
}
progressBar_mc.createEmptyMovieClip("stroke_mc", progressBar_mc.getNextHighestDepth());
with (progressBar_mc.stroke_mc) {
       lineStyle(0, 0x000000);
       moveTo(0, 0);
       lineTo(100, 0);
       lineTo(100, 10);
       lineTo(0, 10);
       lineTo(0, 0);
}

var loaded_interval:Number = setInterval(checkBytesLoaded, 500, stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
       var pctLoaded:Number = Math.round(my_ns.bytesLoaded/my_ns.bytesTotal*100);
       loaded_txt.text = Math.round(my_ns.bytesLoaded/1000)+" of
"+Math.round(my_ns.bytesTotal/1000)+" KB loaded ("+pctLoaded+"%)";
       progressBar_mc.bar_mc._xscale = pctLoaded;
       if (pctLoaded>=100) {
       clearInterval(loaded_interval);
       }
}
```

bytesTotal (NetStream.bytesTotal property)

```
public bytesTotal : Number [read-only]
```

The total size in bytes of the file being loaded into the player.

Availability

Flash Lite 3.0

Example

The following example creates a progress bar using the Drawing API and the `bytesLoaded` and `bytesTotal` properties. The bar displays the progress of the operation as video1.flv is loaded into the video object instance called my_video. A text field called `loaded_txt` is dynamically created to display information about the loading progress as well.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
progressBar_mc.createEmptyMovieClip("bar_mc", progressBar_mc.getNextHighestDepth());
with (progressBar_mc.bar_mc) {
       beginFill(0xFF0000);
       moveTo(0, 0);
       lineTo(100, 0);
       lineTo(100, 10);
       lineTo(0, 10);
       lineTo(0, 0);
       endFill();
       _xscale = 0;
}
progressBar_mc.createEmptyMovieClip("stroke_mc", progressBar_mc.getNextHighestDepth());
with (progressBar_mc.stroke_mc) {
       lineStyle(0, 0x000000);
       moveTo(0, 0);
       lineTo(100, 0);
       lineTo(100, 10);
       lineTo(0, 10);
       lineTo(0, 0);
}

var loaded_interval:Number = setInterval(checkBytesLoaded, 500, stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
       var pctLoaded:Number = Math.round(my_ns.bytesLoaded/my_ns.bytesTotal*100);
       loaded_txt.text = Math.round(my_ns.bytesLoaded/1000)+" of
"+Math.round(my_ns.bytesTotal/1000)+" KB loaded ("+pctLoaded+"%)";
       progressBar_mc.bar_mc._xscale = pctLoaded;
       if (pctLoaded>=100) {
       clearInterval(loaded_interval);
       }
}
```

currentFps (NetStream.currentFps property)

```
public currentFps : Number [read-only]
```

The number of frames per second being displayed. If you are exporting FLV files to be played back on a number of systems, you can check this value during testing to determine how much compression to apply when exporting the file.

Availability

Flash Lite 3.0

Note: *This property is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Example

The following example creates a text field that displays the current number of frames per second that video1.flv displays.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("fps_txt", this.getNextHighestDepth(), 10, 10, 50, 22);
fps_txt.autoSize = true;
var fps_interval:Number = setInterval(displayFPS, 500, stream_ns);
function displayFPS(my_ns:NetStream) {
       fps_txt.text = "currentFps (frames per second): "+Math.floor(my_ns.currentFps);
}
time (NetStream.time property)
```

NetStream constructor

```
public NetStream(connection:"NetConnection" on page 472)
```

Creates a stream that can be used for playing FLV files through the specified NetConnection object.

Note: *This class is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.*

Parameters

connection : `NetConnection` - A NetConnection object.

Example

The following code first constructs a new NetConnection object, `connection_nc`, and uses it to construct a new NetStream object called `stream_ns`. Select New Video from the Library options menu to create a video object instance, and give it an instance name `my_video`.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
```

See also

"NetConnection" on page 472, "attachVideo (Video.attachVideo method)" on page 661

time (NetStream.time property)

```
public time : Number [read-only]
```

The position of the playhead, in seconds.

Availability

Flash Lite 3.0

Note: *This property is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Example

The following example displays the current position of the playhead in a dynamically created text field called `time_txt`. Select New Video from the Library options menu to create a video object instance, and give it an instance name my_video. Create a new video object called my_video. Add the following ActionScript to your FLA or AS file:

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
//
stream_ns.onStatus = function(infoObject:Object) {
       statusCode_txt.text = infoObject.code;
};

this.createTextField("time_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
time_txt.text = "LOADING";

var time_interval:Number = setInterval(checkTime, 500, stream_ns);
function checkTime(my_ns:NetStream) {
       var ns_seconds:Number = my_ns.time;
       var minutes:Number = Math.floor(ns_seconds/60);
       var seconds = Math.floor(ns_seconds%60);
       if (seconds<10) {
       seconds = "0"+seconds;
       }
       time_txt.text = minutes+":"+seconds;
}
```

onStatus (NetStream.onStatus handler)

```
onStatus = function(infoObject:Object) {}
```

Invoked every time a status change or error is posted for the NetStream object. If you want to respond to this event handler, you must create a function to process the information object.

The information object has a code property containing a string that describes the result of the `onStatus` handler, and a `level` property containing a string that is either `status` or `error`.

The following events notify you when certain NetStream activities occur.

Code property	Level property	Meaning
`NetStream.Buffer.Empty`	status	Data is not being received quickly enough to fill the buffer. Data flow will be interrupted until the buffer refills, at which time a `NetStream.Buffer.Full` message will be sent and the stream will begin playing again.
`NetStream.Buffer.Full`	status	The buffer is full and the stream will begin playing.
`NetStream.Buffer.Flush`	status	Data has finished streaming, and the remaining buffer will be emptied.
`NetStream.Play.Start`	status	Playback has started.
`NetStream.Play.Stop`	status	Playback has stopped.

Code property	Level property	Meaning
`NetStream.Play.StreamNotFound`	status	The FLV file passed to the `play()` method can't be found.
`NetStream.Seek.InvalidTime`	error	For video downloaded with progressive download, the user has tried to seek or play past the end of the video data that has downloaded thus far, or past the end of the video once the entire file has downloaded. The `Error.message.details` property contains a time code that indicates the last valid position to which the user can seek. See `Error.message` property.
`NetStream.Seek.Notify`	status	The seek operation is complete.

If you consistently see errors regarding the buffer, you should try changing the buffer using the `NetStream.setBufferTime()` method.

Availability

Flash Lite 3.0

Parameters

infoObject: Object - A parameter defined according to the status message or error message.

Example

The following example displays data about the stream in the Output panel:

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
stream_ns.onStatus = function(infoObject:Object) {
           trace("NetStream.onStatus called: ("+getTimer()+" ms)");
           for (var prop in infoObject) {
               trace("\t"+prop+":\t"+infoObject[prop]);
           }
           trace("");
};
```

onCuePoint (NetStream.onCuePoint handler)

```
onCuePoint = function(infoObject:Object) {}
```

Invoked when an embedded cue point is reached while playing an FLV file. You can use this handler to trigger actions in your code when the video reaches a specific cue point. This lets you synchronize other actions in your application with video playback events.

Two types of cue points can be embedded in an FLV file.

A “navigation” cue point specifies a keyframe within the FLV file and the cue point's `time` property corresponds to that exact keyframe. Navigation cue points are often used as bookmarks or entry points to let users navigate through the video file.

An “event” cue point is specified by time, whether or not that time corresponds to a specific keyframe. An event cue point usually represents a time in the video when something happens that could be used to trigger other application events.

The `onCuePoint` event handler receives an object with these properties:

Property	Description
name	The name given to the cue point when it was embedded in the FLV file.
time	The time in seconds at which the cue point occurred in the video file during playback.
type	The type of cue point that was reached, either "navigation" or "event".
parameters	A associative array of name/value pair strings specified for this cue point. Any valid string can be used for the parameter name or value.

You can define cue points in an FLV file when you first encode the file, or when you import a video clip in the Flash Authoring tool by using the Video Import wizard.

The `onMetaData` event handler also retrieves information about the cue points in a video file. However the `onMetaData` event handler gets information about all of the cue points before the video begins playing. The `onCuePoint` event handler receives information about a single cue point at the time specified for that cue point during playback.

Generally if you want your code to respond to a specific cue point at the time it occurs you should use the `onCuePoint` event handler to trigger some action in your code.

You can use the list of cue points provided to the `onMetaData()` event handler to let your user start playing the video at predefined points along the video stream. Pass the value of the cue point's time property to the `NetStream.seek()` method to play the video from that cue point.

Availability

Flash Lite 3.0

Parameters

infoObject: Object - An object containing the name, time, type, and parameters for the cue point.

Example

The code in this example starts by creating new NetConnection and NetStream objects. Then it defines the `onCuePoint()` handler for the NetStream object. The handler cycles through each named property in the infoObject object and prints the property's name and value. When it finds the property named parameters it cycles through each parameter name in the list and prints the parameter name and value.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
var ns:NetStream = new NetStream(nc);

ns.onCuePoint = function(infoObject:Object)
{
       trace("onCuePoint:");
       for (var propName:String in infoObject) {
           if (propName != "parameters")
           {
               trace(propName + " = " + infoObject[propName]);
           }
           else
           {
               trace("parameters =");
               if (infoObject.parameters != undefined) {
                   for (var paramName:String in infoObject.parameters)
                   {
                       trace(" " + paramName + ": " + infoObject.parameters[paramName]);
                   }
               }
               else
               {
                   trace("undefined");
               }
           }
       }
       trace("---------");
}

ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");
```

This causes the following information to be displayed:

```
onCuePoint:
parameters =
lights: beginning
type = navigation
time = 0.418
name = point1
---------
onCuePoint:
parameters =
lights: middle
type = navigation
time = 7.748
name = point2
---------
onCuePoint:
parameters =
lights: end
type = navigation
time = 16.02
name = point3
---------
```

The parameter name “lights” is an arbitrary name used by the author of the example video. You can give cue point parameters any name you want.

onMetaData (NetStream.onMetaData handler)

```
onMetaData = function(infoObject:Object) {}
```

Invoked when the Flash Lite player receives descriptive information embedded in the FLV file being played.

The Flash Video Exporter utility (version 1.1 or greater) embeds a video's duration, creation date, data rates, and other information into the video file itself. Different video encoders embed different sets of metadata.

This handler is triggered after a call to the `NetStream.play()` method, but before the video playhead has advanced.

In many cases the duration value embedded in FLV metadata approximates the actual duration but is not exact. In other words it will not always match the value of the `NetStream.time` property when the playhead is at the end of the video stream.

Availability

Flash Lite 3.0

Parameters

`infoObject:` Object - An object containing one property for each metadata item.

Example

The code in this example starts by creating new NetConnection and NetStream objects. Then it defines the `onMetaData()` handler for the NetStream object. The handler cycles through each named property in the infoObject object and prints the property's name and value.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
var ns:NetStream = new NetStream(nc);

ns.onMetaData = function(infoObject:Object) {
       for (var propName:String in infoObject) {
           trace(propName + " = " + infoObject[propName]);
       }
};

ns.play("http://www.helpexamples.com/flash/video/water.flv");
```

This causes the following information to be displayed:

```
canSeekToEnd = true
    videocodecid = 4
    framerate = 15
    videodatarate = 400
    height = 215
    width = 320
    duration = 7.347
```

The list of properties will vary depending on the software that was used to encode the FLV file.

close (NetStream.close) method

```
public close() : Void
```

Stops playing all data on the stream, sets the `NetStream.time` property to 0, and makes the stream available for another use. This command also deletes the local copy of an FLV file that was downloaded using HTTP. Although Flash Lite player deletes the local copy of the FLV file that it creates, a copy of the video may persist in the browser's cache directory. If complete prevention of caching or local storage of the FLV file is required, use Flash Media Server.

Availability

Flash Lite 3.0

Note: *This method is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Example

The following `close()` function closes a connection and deletes the temporary copy of video1.flv that was stored on the local disk when you click the button called close_btn:

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

close_btn.onRelease = function(){
       stream_ns.close();
};
```

pause (NetStream.pause) method

```
public pause([flag:Boolean]) : Void
```

Pauses or resumes playback of a stream.

The first time you call this method (without sending a parameter), it pauses play; the next time, it resumes play. You might want to attach this method to a button that the user presses to pause or resume playback.

Availability

Flash Lite 3.0

Note: *This method is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Parameters

`flag : Boolean` [optional] - A Boolean value specifying whether to pause play `(true)` or resume play `(false)`. If you omit this parameter, `NetStream.pause()` acts as a toggle: the first time it is called on a specified stream, it pauses play, and the next time it is called, it resumes play.

Example

The following examples illustrate some uses of this method:

```
my_ns.pause(); // pauses play first time issued
my_ns.pause(); // resumes play
my_ns.pause(false); // no effect, play continues
my_ns.pause(); // pauses play
```

play (NetStream.play method)

```
public play(name:Object, start:Number, len:Number, reset:Object) : Void
```

Begins playback of an external video (FLV) file. To view video data, you must call a `Video.attachVideo()` method; audio being streamed with the video, or an FLV file that contains only audio, is played automatically.

If you want to control the audio associated with an FLV file, you can use `MovieClip.attachAudio()` to route the audio to a movie clip; you can then create a Sound object to control some aspects of the audio. For more information, see `MovieClip.attachAudio()`.

If the FLV file can't be found, the `NetStream.onStatus` event handler is invoked. If you want to stop a stream that is currently playing, use `NetStream.close()`.

You can play local FLV files that are stored in the same directory as the SWF file or in a subdirectory; you can't navigate to a higher-level directory. For example, if the SWF file is located in a directory named /training, and you want to play a video stored in the /training/videos directory, you would use the following syntax:

```
my_ns.play("videos/videoName.flv");
```

To play a video stored in the /training directory, you would use the following syntax:

```
my_ns.play("videoName.flv");
```

When using this method, consider the Flash Player security model.

For Flash Player 8:

`NetStream.play()` is not allowed if the calling SWF file is in the local-with-file-system sandbox and the resource is in a non-local sandbox.

Network sandbox access from the local-trusted or local-with-networking sandbox requires permission from the website via a cross-domain policy file.

For more information, see the following:

The Flash Player 9 Security white paper at http://www.adobe.com/go/fp9_0_security

The Flash Player 8 Security-Related API white paper at http://www.adobe.com/go/fp8_security_apis

Availability

Flash Lite 3.0

Note: *This method is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Parameters

`name` : Object - The name of an FLV file to play, in quotation marks. Both http:// and file:// formats are supported; the file:// location is always relative to the location of the SWF file.

`start` : Number - Use with Flash Media Server; see: the Flash Media Server documentation.

`len` : Number - Use with Flash Media Server; see: the Flash Media Server documentation.

`reset` : Object - Use with Flash Media Server; see: the Flash Media Server documentation.

Example

The following example illustrates some ways to use the `NetStream.play()` method. You can play a file that is on a user's computer. The joe_user directory is a subdirectory of the directory where the SWF is stored. And, you can play a file on a server:

```
// Play a file that is on the user's computer.
my_ns.play("file://joe_user/flash/videos/lectureJune26.flv");

// Play a file on a server.
my_ns.play("http://someServer.someDomain.com/flash/video/orientation.flv");
```

seek (NetStream.seek method)

```
public seek(offset:Number) : Void
```

Seeks the keyframe closest to the specified number of seconds from the beginning of the stream. Playback resumes when this location is reached.

Availability

Flash Lite 3.0

Note: *This method is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Parameters

`offset : Number` - The approximate time value, in seconds, by which to move the playhead in an FLV file. The playhead moves to the keyframe of the video that's closest to the value specified by `offset`.

- To return to the beginning of the stream, pass 0 to `offset`.
- To seek forward from the beginning of the stream, pass the number of seconds you want to advance. For example, to position the playhead 15 seconds from the beginning, use *my_ns*`.seek(15)`.
- To seek relative to the current position, pass *my_ns*`.time + n` or *my_ns*`.time - n` to seek *n* seconds forward or backward, respectively, from the current position. For example, to rewind 20 seconds from the current position, use *my_ns*`.seek(`*my_ns*`.time - 20)`.

The precise location to which the playhead moves differs according to the frames-per-second (fps) setting at which the video was exported. For example, suppose you have two video objects that represent the same video, one exported at 6 fps and the other at 30 fps. If you then use `my_ns.seek(15)` for both objects, the playhead moves to two different locations.

Example

The following example illustrates some ways to use the `NetStream.seek()` command. You can return to the beginning of the stream, move to a location 30 seconds from the beginning of the stream, and move backward three minutes from the current location:

```
// Return to the beginning of the stream
my_ns.seek(0);

// Move to a location 30 seconds from the beginning of the stream
my_ns.seek(30);

// Move backwards three minutes from current location
my_ns.seek(my_ns.time - 180);
```

setBufferTime (NetStream.setBufferTime) method

```
public setBufferTime(bufferTime:Number) : Void
```

Specifies how long to buffer messages before starting to display the stream. For example, if you want to make sure that the first 15 seconds of the stream play without interruption, set `bufferTime` to 15; Flash begins playing the stream only after 15 seconds of data are buffered.

Availability

Flash Lite 3.0

Note: *This method is also supported in Flash Player 6 when used with Flash Media Server. For more information, see the Flash Media Server documentation.*

Parameters

`bufferTime` : Number - The time, in seconds, during which data is buffered before Flash begins displaying data. The default value is 0.1 (one-tenth of a second).

Example

See the example for `NetStream.bufferLength`.

Number

```
Object
    |
    +-Number
```

```
public class Number
extends Object
```

The Number class is a simple wrapper object for the Number data type. You can manipulate primitive numeric values by using the methods and properties associated with the Number class. This class is identical to the JavaScript Number class.

The properties of the Number class are static, which means you do not need an object to use them, so you do not need to use the constructor.

The following example calls the `toString()` method of the Number class, which returns the string `1234`:

```
var myNumber:Number = new Number(1234);
myNumber.toString();
```

The following example assigns the value of the `MIN_VALUE` property to a variable declared without the use of the constructor:

```
var smallest:Number = Number.MIN_VALUE;
```

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
static	MAX_VALUE : Number	The largest representable number (double-precision IEEE-754).
static	MIN_VALUE : Number	The smallest representable number (double-precision IEEE-754).
static	NaN : Number	The IEEE-754 value representing Not A Number (NaN).
static	NEGATIVE_INFINITY : Number	Specifies the IEEE-754 value representing negative infinity.
static	POSITIVE_INFINITY : Number	Specifies the IEEE-754 value representing positive infinity.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Constructor summary

Signature	Description
Number (num : Object)	Creates a new Number object.

Method summary

Modifiers	Signature	Description
	toString (radix : Number) : String	Returns the string representation of the specified Number object (*myNumber*).
	valueOf () : Number	Returns the primitive value type of the specified Number object.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

MAX_VALUE (Number.MAX_VALUE property)

```
public static MAX_VALUE : Number
```

The largest representable number (double-precision IEEE-754). This number is approximately 1.79e+308.

Availability

Flash Lite 2.0

Example

The following ActionScript displays the largest and smallest representable numbers to the Output panel.

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

This code displays the following values:

```
Number.MIN_VALUE = 4.94065645841247e-324
Number.MAX_VALUE = 1.79769313486232e+308
```

MIN_VALUE (Number.MIN_VALUE property)

```
public static MIN_VALUE : Number
```

The smallest representable number (double-precision IEEE-754). This number is approximately 5e-324.

Availability

Flash Lite 2.0

Example

The following ActionScript displays the largest and smallest representable numbers to the Output panel to the log file.

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

This code displays the following values:

```
Number.MIN_VALUE = 4.94065645841247e-324
Number.MAX_VALUE = 1.79769313486232e+308
```

NaN (Number.NaN property)

```
public static NaN : Number
```

The IEEE-754 value representing Not A Number (`NaN`).

Availability

Flash Lite 2.0

See also

`isNaN function`

NEGATIVE_INFINITY (Number.NEGATIVE_INFINITY property)

```
public static NEGATIVE_INFINITY : Number
```

Specifies the IEEE-754 value representing negative infinity. The value of this property is the same as that of the constant `-Infinity`.

Negative infinity is a special numeric value that is returned when a mathematical operation or function returns a negative value larger than can be represented.

Availability

Flash Lite 2.0

Example

This example compares the result of dividing the following values.

```
var posResult:Number = 1/0;
if (posResult == Number.POSITIVE_INFINITY) {
    trace("posResult = "+posResult); // output: posResult = Infinity
}
var negResult:Number = -1/0;
if (negResult == Number.NEGATIVE_INFINITY) {
    trace("negResult = "+negResult); // output: negResult = -Infinity
```

Number constructor

```
public Number(num:Object)
```

Creates a new Number object. The `new Number` constructor is primarily used as a placeholder. A Number object is not the same as the `Number()` function that converts a parameter to a primitive value.

Availability

Flash Lite 2.0

Parameters

`num` : Object - The numeric value of the Number object being created or a value to be converted to a number. The default value is 0 if `value` is not provided.

Example

The following code constructs new Number objects:

```
var n1:Number = new Number(3.4);
var n2:Number = new Number(-10);
```

See also

`toString (Number.toString method)`, `valueOf (Number.valueOf method)`

POSITIVE_INFINITY (Number.POSITIVE_INFINITY property)

```
public static POSITIVE_INFINITY : Number
```

Specifies the IEEE-754 value representing positive infinity. The value of this property is the same as that of the constant `Infinity`.

Positive infinity is a special numeric value that is returned when a mathematical operation or function returns a value larger than can be represented.

Availability

Flash Lite 2.0

Example

This example compares the result of dividing the following values.

```
var posResult:Number = 1/0;
if (posResult == Number.POSITIVE_INFINITY) {
    trace("posResult = "+posResult); // output: posResult = Infinity
}
var negResult:Number = -1/0;
if (negResult == Number.NEGATIVE_INFINITY) {
    trace("negResult = "+negResult); // output: negResult = -Infinity
```

toString (Number.toString method)

```
public toString(radix:Number) : String
```

Returns the string representation of the specified Number object (*myNumber*).

Availability

Flash Lite 2.0

Parameters

`radix` : Number - Specifies the numeric base (from 2 to 36) to use for the number-to-string conversion. If you do not specify the `radix` parameter, the default value is 10.

Returns

String - A string.

Example

The following example uses 2 and 8 for the `radix` parameter and returns a string that contains the corresponding representation of the number 9:

```
var myNumber:Number = new Number(9);
trace(myNumber.toString(2)); // output: 1001
trace(myNumber.toString(8)); // output: 11
```

The following example results in a hexadecimal value.

```
var r:Number = new Number(250);
var g:Number = new Number(128);
var b:Number = new Number(114);
var rgb:String = "0x"+ r.toString(16)+g.toString(16)+b.toString(16);
trace(rgb);
// output: rgb:0xFA8072 (Hexadecimal equivalent of the color 'salmon')
```

valueOf (Number.valueOf method)

```
public valueOf() : Number
```

Returns the primitive value type of the specified Number object.

Availability

Flash Lite 2.0

Returns

Number - A string.

Example

The following example results in the primitive value of the numSocks object.

```
var numSocks = new Number(2);
trace(numSocks.valueOf()); // output: 2
```

Object

`Object`

`public class Object`

The Object class is at the root of the ActionScript class hierarchy. This class contains a small subset of the features provided by the JavaScript Object class.

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
	constructor : Object	Reference to the constructor function for a given object instance.
	__proto__ : Object	Refers to the `prototype` property of the class (ActionScript 2.0) or constructor function (ActionScript 1.0) used to create the object.
`static`	prototype : Object	A reference to the superclass of a class or function object.
	__resolve : Object	A reference to a user-defined function that is invoked if ActionScript code refers to an undefined property or method.

Constructor summary

Signature	Description
Object `()`	Creates an Object object and stores a reference to the object's constructor method in the object's `constructor` property.

Method summary

Modifiers	Signature	Description
	addProperty `(name:` String `, getter:` Function `, setter:` Function `) :` Boolean	Creates a getter/setter property.
	hasOwnProperty `(name:` String `) :` Boolean	Indicates whether an object has a specified property defined.
	isPropertyEnumerable `(name:` String `) :` Boolean	Indicates whether the specified property exists and is enumerable.
	isPrototypeOf `(theClass:` Object `) :` Boolean	Indicates whether an instance of the Object class is in the prototype chain of the object specified as an argument.

Modifiers	Signature	Description
static	registerClass(name:String, theClass:Function) : Boolean	Associates a movie clip symbol with an ActionScript object class.
	toString() : String	Converts the specified object to a string and returns it.
	unwatch(name:String) : Boolean	Removes a watchpoint that Object.watch() created.
	valueOf() : Object	Returns the primitive value of the specified object.
	watch(name:String, callback:Function, [userData:Object]) : Boolean	Registers an event handler to be invoked when a specified property of an ActionScript object changes.

addProperty (Object.addProperty method)

```
public addProperty(name:String, getter:Function, setter:Function) : Boolean
```

Creates a getter/setter property. When Flash reads a getter/setter property, it invokes the `get` function, and the function's return value becomes the value of `name`. When Flash writes a getter/setter property, it invokes the `set` function and passes it the new value as a parameter. If a property with the given name already exists, the new property overwrites it.

A "get" function is a function with no parameters. Its return value can be of any type. Its type can change between invocations. The return value is treated as the current value of the property.

A "set" function is a function that takes one parameter, which is the new value of the property. For example, if property `x` is assigned by the statement `x = 1`, the set function is passed the parameter `1` of type number. The return value of the set function is ignored.

You can add getter/setter properties to prototype objects. If you add a getter/setter property to a prototype object, all object instances that inherit the prototype object inherit the getter/setter property. This makes it possible to add a getter/setter property in one location, the prototype object, and have it propagate to all instances of a class (similar to adding methods to prototype objects). If a get/set function is invoked for a getter/setter property in an inherited prototype object, the reference passed to the get/set function is the originally referenced object--not the prototype object.

If invoked incorrectly, `Object.addProperty()` can fail with an error. The following table describes errors that can occur:

Error condition	What happens
`name` is not a valid property name; for example, an empty string.	Returns `false` and the property is not added.
`getter` is not a valid function object.	Returns `false` and the property is not added.
`setter` is not a valid function object.	Returns `false` and the property is not added.

Availability

Flash Lite 2.0

Parameters

name : String - A string; the name of the object property to create.

getter : Function - The function that is invoked to retrieve the value of the property; this parameter is a Function object.

setter : Function - The function that is invoked to set the value of the property; this parameter is a Function object. If you pass the value `null` for this parameter, the property is read-only.

Returns

Boolean - A Boolean value: `true` if the property is successfully created; `false` otherwise.

Example

In the following example, an object has two internal methods, `setQuantity()` and `getQuantity()`. A property, `bookcount`, can be used to invoke these methods when it is either set or retrieved. A third internal method, `getTitle()`, returns a read-only value that is associated with the property `bookname`. When a script retrieves the value of `myBook.bookcount`, the ActionScript interpreter automatically invokes `myBook.getQuantity()`. When a script modifies the value of `myBook.bookcount`, the interpreter invokes `myObject.setQuantity()`. The `bookname` property does not specify a `set` function, so attempts to modify `bookname` are ignored.

```
function Book() {
    this.setQuantity = function(numBooks:Number):Void {
    this.books = numBooks;
    };
    this.getQuantity = function():Number {
    return this.books;
    };
    this.getTitle = function():String {
    return "Catcher in the Rye";
    };
    this.addProperty("bookcount", this.getQuantity, this.setQuantity);
    this.addProperty("bookname", this.getTitle, null);
}
var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
// output: You ordered 5 copies of Catcher in the Rye
```

The previous example works, but the properties `bookcount` and `bookname` are added to every instance of the Book object, which requires having two properties for every instance of the object. If there are many properties, such as `bookcount` and `bookname`, in a class, they could consume a great deal of memory. Instead, you can add the properties to `Book.prototype` so that the `bookcount` and `bookname` properties exist only in one place. The effect, however, is the same as that of the code in the example that added `bookcount` and `bookname` directly to every instance. If an attempt is made to access either property in a Book instance, the property's absence will cause the prototype chain to be ascended until the versions defined in `Book.prototype` are encountered. The following example shows how to add the properties to `Book.prototype`:

```
function Book() {}
Book.prototype.setQuantity = function(numBooks:Number):Void {
    this.books = numBooks;
};
Book.prototype.getQuantity = function():Number {
    return this.books;
};
Book.prototype.getTitle = function():String {
    return "Catcher in the Rye";
};
Book.prototype.addProperty("bookcount", Book.prototype.getQuantity,
Book.prototype.setQuantity);
Book.prototype.addProperty("bookname", Book.prototype.getTitle, null);
var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
```

The following example shows how to use the implicit getter and setter functions available in ActionScript 2.0. Rather than defining the `Book` function and editing `Book.prototype`, you define the Book class in an external file named Book.as. The following code must be in a separate external file named Book.as that contains only this class definition and resides within the Flash application's classpath:

```
class Book {
    var books:Number;
    function set bookcount(numBooks:Number):Void {
    this.books = numBooks;
    }
    function get bookcount():Number {
    return this.books;
    }
    function get bookname():String {
    return "Catcher in the Rye";
    }
}
```

The following code can then be placed in a FLA file and will function the same way as it does in the previous examples:

```
var myBook:Book = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
```

See also

`getProperty function`, `setInterval function`

constructor (Object.constructor property)

```
public constructor : Object
```

Reference to the constructor function for a given object instance. The `constructor` property is automatically assigned to all objects when they are created using the constructor for the Object class.

Availability

Flash Lite 2.0

Example

The following example is a reference to the constructor function for the myObject object.

```
var my_str:String = new String("sven");
trace(my_str.constructor == String); //output: true
```

If you use the `instanceof` operator, you can also determine if an object belongs to a specified class:

```
var my_str:String = new String("sven");
trace(my_str instanceof String); //output: true
```

However, in the following example the `Object.constructor` property converts primitive data types (such as the string literal seen here) into wrapper objects. The `instanceof` operator does not perform any conversion, as seen in the following example:

```
var my_str:String = "sven";
trace(my_str.constructor == String); //output: true
trace(my_str instanceof String); //output: false
```

See also

`instanceof operator`

hasOwnProperty (Object.hasOwnProperty method)

```
public hasOwnProperty(name:String) : Boolean
```

Indicates whether an object has a specified property defined. This method returns `true` if the target object has a property that matches the string specified by the `name` parameter, and `false` otherwise. This method does not check the object's prototype chain and returns `true` only if the property exists on the object itself.

Availability

Flash Lite 2.0

Parameters

name : String - A string; the name of the property.

Returns

Boolean - A Boolean value: `true` if the target object has the property specified by the `name` parameter, `false` otherwise.

isPropertyEnumerable (Object.isPropertyEnumerable method)

```
public isPropertyEnumerable(name:String) : Boolean
```

Indicates whether the specified property exists and is enumerable. If `true`, then the property exists and can be enumerated in a `for..in` loop. The property must exist on the target object because this method does not check the target object's prototype chain.

Properties that you create are enumerable, but built-in properties are generally not enumerable.

Availability

Flash Lite 2.0

Parameters

name : String - The name of the property to check for.

Returns

Boolean - A Boolean value: `true` if the property specified by the `name` parameter is enumerable.

Example

The following example creates a generic object, adds a property to the object, then checks whether the object is enumerable. By way of contrast, the example also shows that a built-in property, the `Array.length` property, is not enumerable.

```
var myObj:Object = new Object();
myObj.prop1 = "hello";
trace(myObj.isPropertyEnumerable("prop1")); // Output: true

var myArray = new Array();
trace(myArray.isPropertyEnumerable("length")); // Output: false
```

See also

`for..in statement`

isPrototypeOf (Object.isPrototypeOf method)

```
public isPrototypeOf(theClass:Object) : Boolean
```

Indicates whether an instance of the Object class is in the prototype chain of the object specified as an argument. This method returns `true` if the object is in the prototype chain of the object specified by the `theClass` parameter. The method returns `false` not only if the target object is absent from the prototype chain of the theClass object, but also if the `theClass` argument is not an object.

Availability

Flash Lite 2.0

Parameters

theClass : Object - The class in whose prototype chain to check for the object.

Returns

Boolean - A Boolean value: `true` if the object is in the prototype chain of the object specified by the `theClass` parameter; `false` otherwise.

Object constructor

```
public Object()
```

Creates an Object object and stores a reference to the object's constructor method in the object's `constructor` property.

Availability

Flash Lite 2.0

Example

The following example creates a generic object named `myObject`:

```
var myObject:Object = new Object();
```

__proto__ (Object.__proto__ property)

```
public __proto__ : Object
```

Refers to the `prototype` property of the class (ActionScript 2.0) or constructor function (ActionScript 1.0) used to create the object. The `__proto__` property is automatically assigned to all objects when they are created. The ActionScript interpreter uses the `__proto__` property to access the `prototype` property of the object's class or constructor function to find out what properties and methods the object inherits from its superclass.

Availability

Flash Lite 2.0

Example

The following example creates a class named Shape and a subclass of Shape named Circle.

```
// Shape class defined in external file named Shape.as
class Shape {
    function Shape() {}
}

// Circle class defined in external file named Circle.as
class Circle extends Shape{
    function Circle() {}
}
```

The Circle class can be used to create two instances of Circle:

```
var oneCircle:Circle = new Circle();
var twoCircle:Circle = new Circle();
```

The following trace statements show that the `__proto_` property of both instances refers to the `prototype` property of the Circle class.

```
trace(Circle.prototype == oneCircle.__proto__); // Output: true
trace(Circle.prototype == twoCircle.__proto__); // Output: true
```

See also

`prototype (Object.prototype property)`

prototype (Object.prototype property)

```
public static prototype : Object
```

A reference to the superclass of a class or function object. The `prototype` property is automatically created and attached to any class or function object you create. This property is static in that it is specific to the class or function you create. For example, if you create a custom class, the value of the `prototype` property is shared by all instances of the class, and is accessible only as a class property. Instances of your custom class cannot directly access the `prototype` property, but can access it through the `__proto__` property.

Availability

Flash Lite 2.0

Example

The following example creates a class named Shape and a subclass of Shape named Circle.

```
// Shape class defined in external file named Shape.as
class Shape {
    function Shape() {}
}

// Circle class defined in external file named Circle.as
class Circle extends Shape{
    function Circle() {}
}
```

The Circle class can be used to create two instances of Circle:

```
var oneCircle:Circle = new Circle();
var twoCircle:Circle = new Circle();
```

The following trace statement shows that the `prototype` property of the Circle class points to its superclass Shape. The identifier `Shape` refers to the constructor function of the Shape class.

```
trace(Circle.prototype.constructor == Shape); // Output: true
```

The following trace statement shows how you can use the `prototype` property and the `__proto__` property together to move two levels up the inheritance hierarchy (or prototype chain). The `Circle.prototype.__proto__` property contains a reference to the superclass of the Shape class.

```
trace(Circle.prototype.__proto__ == Shape.prototype); // Output: true
```

See also

`__proto__ (Object.__proto__ property)`

registerClass (Object.registerClass method)

```
public static registerClass(name:String, theClass:Function) : Boolean
```

Associates a movie clip symbol with an ActionScript object class. If a symbol doesn't exist, Flash creates an association between a string identifier and an object class.

When an instance of the specified movie clip symbol is placed on the Timeline, it is registered to the class specified by the `theClass` parameter rather than to the class MovieClip.

When an instance of the specified movie clip symbol is created by using `MovieClip.attachMovie()` or `MovieClip.duplicateMovieClip()`, it is registered to the class specified by `theClass` rather than to the MovieClip class. If `theClass` is `null`, this method removes any ActionScript class definition associated with the specified movie clip symbol or class identifier. For movie clip symbols, any existing instances of the movie clip remain unchanged, but new instances of the symbol are associated with the default class MovieClip.

If a symbol is already registered to a class, this method replaces it with the new registration.

When a movie clip instance is placed by the Timeline or created using `attachMovie()` or `duplicateMovieClip()`, ActionScript invokes the constructor for the appropriate class with the keyword `this` pointing to the object. The constructor function is invoked with no parameters.

If you use this method to register a movie clip with an ActionScript class other than MovieClip, the movie clip symbol doesn't inherit the methods, properties, and events of the built-in MovieClip class unless you include the MovieClip class in the prototype chain of the new class. The following code creates a new ActionScript class called `theClass` that inherits the properties of the MovieClip class:

```
theClass.prototype = new MovieClip();
```

Availability
Flash Lite 2.0

Parameters
name : String - String; the linkage identifier of the movie clip symbol or the string identifier for the ActionScript class.

theClass : Function - A reference to the constructor function of the ActionScript class or `null` to unregister the symbol.

Returns
Boolean - A Boolean value: if the class registration succeeds, a value of `true` is returned; `false` otherwise.

See also
`attachMovie (MovieClip.attachMovie method)`, `duplicateMovieClip (MovieClip.duplicateMovieClip method)`

__resolve (Object.__resolve property)

`public __resolve : Object`

A reference to a user-defined function that is invoked if ActionScript code refers to an undefined property or method. If ActionScript code refers to an undefined property or method of an object, Flash Lite player determines whether the object's `__resolve` property is defined. If `__resolve` is defined, the function to which it refers is executed and passed the name of the undefined property or method. This lets you programmatically supply values for undefined properties and statements for undefined methods and make it seem as if the properties or methods are actually defined. This property is useful for enabling highly transparent client/server communication, and is the recommended way of invoking server-side methods.

Availability
Flash Lite 2.0

Example
The following examples progressively build upon the first example and illustrate five different usages of the `__resolve` property. To aid understanding, key statements that differ from the previous usage are in bold typeface.

Usage 1: the following example uses `__resolve` to build an object where every undefined property returns the value `"Hello, world!"`.

```
// instantiate a new object
var myObject:Object = new Object();

// define the __resolve function
myObject.__resolve = function (name) {
    return "Hello, world!";
};
trace (myObject.property1); // output: Hello, world!
trace (myObject.property2); // output: Hello, world!
```

Usage 2: the following example uses `__resolve` as a *functor*, which is a function that generates functions. Using `__resolve` redirects undefined method calls to a generic function named `myFunction`.

```
// instantiate a new object
var myObject:Object = new Object();

// define a function for __resolve to call
myObject.myFunction = function (name) {
    trace("Method " + name + " was called");
};

// define the __resolve function
myObject.__resolve = function (name) {
     return function () { this.myFunction(name); };
};

// test __resolve using undefined method names
myObject.someMethod(); // output: Method someMethod was called
myObject.someOtherMethod(); //output: Method someOtherMethod was called
```

Usage 3: The following example builds on the previous example by adding the ability to cache resolved methods. By caching methods, `__resolve` is called only once for each method of interest. This allows *lazy construction* of object methods. Lazy construction is an optimization technique that defers the creation, or *construction*, of methods until the time at which a method is first used.

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
    trace("Method "+name+" was called");
};
// define the __resolve function
myObject.__resolve = function(name) {
    trace("Resolve called for "+name); // to check when __resolve is called
    // Not only call the function, but also save a reference to it
    var f:Function = function () {
        this.myFunction(name);
    };
    // create a new object method and assign it the reference
    this[name] = f;
    // return the reference
    return f;
};
// test __resolve using undefined method names
// __resolve will only be called once for each method name
myObject.someMethod(); // calls __resolve
myObject.someMethod(); // does not call __resolve because it is now defined
myObject.someOtherMethod(); // calls __resolve
myObject.someOtherMethod(); // does not call __resolve, no longer undefined
```

Usage 4: The following example builds on the previous example by reserving a method name, `onStatus()`, for local use so that it is not resolved in the same way as other undefined properties. Added code is in bold typeface.

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
    trace("Method "+name+" was called");
};
// define the __resolve function
myObject.__resolve = function(name) {
    // reserve the name "onStatus" for local use
    if (name == "onStatus") {
        return undefined;
    }
    trace("Resolve called for "+name); // to check when __resolve is called
    // Not only call the function, but also save a reference to it
    var f:Function = function () {
        this.myFunction(name);
    };
    // create a new object method and assign it the reference
    this[name] = f;
    // return the reference
    return f;
};
// test __resolve using the method name "onStatus"
trace(myObject.onStatus("hello"));
// output: undefined
```

Usage 5: The following example builds on the previous example by creating a functor that accepts parameters. This example makes extensive use of the arguments object, and uses several methods of the Array class.

```
// instantiate a new object
var myObject:Object = new Object();

// define a generic function for __resolve to call
myObject.myFunction = function (name) {
    arguments.shift();
    trace("Method " + name + " was called with arguments: " + arguments.join(','));
};

// define the __resolve function
myObject.__resolve = function (name) {
    // reserve the name "onStatus" for local use
    if (name == "onStatus") {
        return undefined;
    }
    var f:Function = function () {
        arguments.unshift(name);
        this.myFunction.apply(this, arguments);
    };
    // create a new object method and assign it the reference
    this[name] = f;
    // return the reference to the function
    return f;
};

// test __resolve using undefined method names with parameters
myObject.someMethod("hello");
// output: Method someMethod was called with arguments: hello

myObject.someOtherMethod("hello","world");
// output: Method someOtherMethod was called with arguments: hello,world
```

See also

`arguments`, `Array`

toString (Object.toString method)

```
public toString() : String
```

Converts the specified object to a string and returns it.

Availability

Flash Lite 2.0

Returns

String - A string.

Example

This example shows the return value for `toString()` on a generic object:

```
var myObject:Object = new Object();
trace(myObject.toString()); // output: [object Object]
```

This method can be overridden to return a more meaningful value. The following examples show that this method has been overridden for the built-in classes Date, Array, and Number:

```
// Date.toString() returns the current date and time
var myDate:Date = new Date();
trace(myDate.toString()); // output: [current date and time]

// Array.toString() returns the array contents as a comma-delimited string
var myArray:Array = new Array("one", "two");
trace(myArray.toString()); // output: one,two

// Number.toString() returns the number value as a string
// Because trace() won't tell us whether the value is a string or number
// we will also use typeof() to test whether toString() works.
var myNumber:Number = 5;
trace(typeof (myNumber)); // output: number
trace(myNumber.toString()); // output: 5
trace(typeof (myNumber.toString())); // output: string
```

The following example shows how to override `toString()` in a custom class. First create a text file named *Vehicle.as* that contains only the Vehicle class definition and place it into your Classes folder inside your Configuration folder.

```
// contents of Vehicle.as
class Vehicle {
    var numDoors:Number;
    var color:String;
    function Vehicle(param_numDoors:Number, param_color:String) {
    this.numDoors = param_numDoors;
    this.color = param_color;
    }
    function toString():String {
    var doors:String = "door";
    if (this.numDoors > 1) {
        doors += "s";
    }
    return ("A vehicle that is " + this.color + " and has " + this.numDoors + " " + doors);
    }
}

// code to place into a FLA file
var myVehicle:Vehicle = new Vehicle(2, "red");
trace(myVehicle.toString());
// output: A vehicle that is red and has 2 doors

// for comparison purposes, this is a call to valueOf()
// there is no primitive value of myVehicle, so the object is returned
// giving the same output as toString().
trace(myVehicle.valueOf());
// output: A vehicle that is red and has 2 doors
```

unwatch (Object.unwatch method)

```
public unwatch(name:String) : Boolean
```

Removes a watchpoint that `Object.watch()` created. This method returns a value of `true` if the watchpoint is successfully removed, `false` otherwise.

Availability

Flash Lite 2.0

Parameters

name : String - A string; the name of the object property that should no longer be watched.

Returns

Boolean - A Boolean value: `true` if the watchpoint is successfully removed, `false` otherwise.

Example

See the example for `Object.watch()`.

See also

`watch (Object.watch method)`, `addProperty (Object.addProperty method)`

valueOf (Object.valueOf method)

```
public valueOf() : Object
```

Returns the primitive value of the specified object. If the object does not have a primitive value, the object is returned.

Availability

Flash Lite 2.0

Returns

Object - The primitive value of the specified object or the object itself.

Example

The following example shows the return value of `valueOf()` for a generic object (which does not have a primitive value) and compares it to the return value of `toString()`. First, create a generic object. Second, create a new Date object set to February 1, 2004, 8:15 AM. The `toString()` method returns the current time in human-readable form. The `valueOf()` method returns the primitive value in milliseconds. Third, create a new Array object containing two simple elements. Both `toString()` and `valueOf()` return the same value: one,two:

```
// Create a generic object
var myObject:Object = new Object();
trace(myObject.valueOf()); // output: [object Object]
trace(myObject.toString()); // output: [object Object]
```

The following examples show the return values for the built-in classes Date and Array, and compares them to the return values of `Object.toString()`:

```
// Create a new Date object set to February 1, 2004, 8:15 AM
// The toString() method returns the current time in human-readable form
// The valueOf() method returns the primitive value in milliseconds
var myDate:Date = new Date(2004,01,01,8,15);
trace(myDate.toString()); // output: Sun Feb 1 08:15:00 GMT-0800 2004
trace(myDate.valueOf()); // output: 1075652100000

// Create a new Array object containing two simple elements
// In this case both toString() and valueOf() return the same value: one,two
var myArray:Array = new Array("one", "two");
trace(myArray.toString()); // output: one,two
trace(myArray.valueOf()); // output: one,two
```

See the example for `Object.toString()` for an example of the return value of `Object.valueOf()` for a custom class that overrides `toString()`.

See also

`toString (Object.toString method)`

watch (Object.watch method)

```
public watch(name:String, callback:Function, [userData:Object]) : Boolean
```

Registers an event handler to be invoked when a specified property of an ActionScript object changes. When the property changes, the event handler is invoked with `myObject` as the containing object.

You can use the `return` statement in your `callback` method definition to affect the value of the property you are watching. The value returned by your `callback` method is assigned to the watched object property. The value you choose to return depends on whether you wish to monitor, modify or prevent changes to the property:

- If you are merely monitoring the property, return the `newVal` parameter.
- If you are modifying the value of the property, return your own value.
- If you want to prevent changes to the property, return the `oldVal` parameter.

If the `callback` method you define does not have a `return` statement, then the watched object property is assigned a value of `undefined`.

A watchpoint can filter (or nullify) the value assignment, by returning a modified `newval` (or `oldval`). If you delete a property for which a watchpoint has been set, that watchpoint does not disappear. If you later recreate the property, the watchpoint is still in effect. To remove a watchpoint, use the `Object.unwatch` method.

Only a single watchpoint can be registered on a property. Subsequent calls to `Object.watch()` on the same property replace the original watchpoint.

The `Object.watch()` method behaves similarly to the `Object.watch()` function in JavaScript 1.2 and later. The primary difference is the `userData` parameter, which is a Flash addition to `Object.watch()` that Netscape Navigator does not support. You can pass the `userData` parameter to the event handler and use it in the event handler.

The `Object.watch()` method cannot watch getter/setter properties. Getter/setter properties operate through *lazy evaluation* – the value of the property is not determined until the property is actually queried. Lazy evaluation is often efficient because the property is not constantly updated; it is, rather, evaluated when needed. However, `Object.watch()` needs to evaluate a property to determine whether to invoke the `callback` function. To work with a getter/setter property, `Object.watch()` needs to evaluate the property constantly, which is inefficient.

Generally, predefined ActionScript properties, such as `_x`, `_y`, `_width`, and `_height`, are getter/setter properties and cannot be watched with `Object.watch()`.

Availability

Flash Lite 2.0

Parameters

name : String - A string; the name of the object property to watch.

callback : Function - The function to invoke when the watched property changes. This parameter is a function object, not a function name as a string. The form of `callback` is `callback(prop, oldVal, newVal, userData)`.

userData : Object [optional] - An arbitrary piece of ActionScript data that is passed to the `callback` method. If the `userData` parameter is omitted, `undefined` is passed to the callback method.

Returns

Boolean - A Boolean value: `true` if the watchpoint is created successfully, `false` otherwise.

Example

The following example uses `watch()` to check whether the `speed` property exceeds the speed limit:

```
// Create a new object
var myObject:Object = new Object();

// Add a property that tracks speed
myObject.speed = 0;

// Write the callback function to be executed if the speed property changes
var speedWatcher:Function = function(prop, oldVal, newVal, speedLimit) {
    // Check whether speed is above the limit
    if (newVal > speedLimit) {
    trace ("You are speeding.");
    }
    else {
    trace ("You are not speeding.");
    }

    // Return the value of newVal.
    return newVal;
}
// Use watch() to register the event handler, passing as parameters:
// - the name of the property to watch: "speed"
// - a reference to the callback function speedWatcher
// - the speedLimit of 55 as the userData parameter
myObject.watch("speed", speedWatcher, 55);

// set the speed property to 54, then to 57
myObject.speed = 54; // output: You are not speeding
myObject.speed = 57; // output: You are speeding

// unwatch the object
myObject.unwatch("speed");
myObject.speed = 54; // there should be no output
```

See also

addProperty (Object.addProperty method), unwatch (Object.unwatch method)

Point (flash.geom.Point)

```
Object
    |
    +-flash.geom.Point

public class Point
extends Object
```

The Point class represents a location in a two-dimensional coordinate system, where *x* represents the horizontal axis and *y* represents the vertical axis.

The following code creates a point at (0,0):

```
var myPoint:Point = new Point();
```

Availability

Flash Lite 3.1

Property summary

Modifiers	Property	Description
	length : Number	The length of the line segment from (0,0) to this point.
	x : Number	The horizontal coordinate of the point.
	y : Number	The vertical coordinate of the point.

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Constructor summary

Signature	Description
Point (x : Number, y : Number)	Creates a new point.

Method summary

Modifiers	Signature	Description
	add (v : Point) : Point	Adds the coordinates of another point to the coordinates of this point to create a new point.
	clone () : Point	Creates a copy of this Point object.
static	distance (pt1 : Point, pt2 : Point) : Number	Returns the distance between pt1 and pt2.
	equals (toCompare : Object) : Boolean	Determines whether two points are equal.
static	interpolate (pt1 : Point, pt2 : Point , f : Number) : Point	Determines a point between two specified points.
	normalize (length : Number) : Void	Scales the line segment between (0,0) and the current point to a set length.
	offset (dx : Number, dy : Number) : Void	Offsets the Point object by the specified amount.

Modifiers	Signature	Description
`static`	polar(len:Number, angle:Number) : Point	Converts a pair of polar coordinates to a Cartesian point coordinate.
	subtract(v:Point : Point	Subtracts the coordinates of another point from the coordinates of this point to create a new point.
	toString (Point.toString method)() : String	Returns a string that contains the values of the *x* and *y* coordinates.

addProperty (Object.addProperty method),hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method),isPrototypeOf (Object.isPrototypeOf method)registerClass (Object.registerClass method),toString (Object.toString method),unwatch (Object.unwatch method),valueOf (Object.valueOf method),watch (Object.watch method)

add (Point.add method)

```
public add(v:Point) : Point
```

Adds the coordinates of another point to the coordinates of this point to create a new point.

Availability

Flash Lite 3.1

Parameters

v: Point - The point to be added.

Returns

Point - The new point.

Example

The following example creates a Point object `resultPoint` by adding `point_2` to `point_1`.

```
import flash.geom.Point;
var point_1:Point = new Point(4, 8);
var point_2:Point = new Point(1, 2);
var resultPoint:Point = point_1.add(point_2);
trace(resultPoint.toString()); // (x=5, y=10)
```

clone (Point.clone method)

```
public clone() : Point
```

Creates a copy of this Point object.

Availability

Flash Lite 3.1

Returns

Point - The new Point object.

Example

The following example creates a copy of the Point object called `clonedPoint` from the values found in the myPoint object. The clonedPoint object contains all of the values from `myPoint`, but it is not the same object.

```
import flash.geom.Point;
var myPoint:Point = new Point(1, 2);
var clonedPoint:Point = myPoint.clone();
trace(clonedPoint.x); // 1
trace(clonedPoint.y); // 2
trace(myPoint.equals(clonedPoint)); // true
trace(myPoint === clonedPoint); // false
```

distance (Point.distance method)

```
public static distance(pt1:Point, pt2:Point)) : Number
```

Returns the distance between pt1 and pt2.

Availability

Flash Lite 3.1

Parameters

pt1 : Point - The first point.

pt2 : Point - The second point.

Returns

Number - The distance between the first and second points.

Example

The following example creates `point_1` and `point_2`, then determines the distance between them (`distanceBetween`).

```
import flash.geom.Point;
var point_1:Point = new Point(-5, 0);
var point_2:Point = new Point(5, 0);
var distanceBetween:Number = Point.distance(point_1, point_2);
trace(distanceBetween); // 10
```

equals (Point.equals method)

```
public equals(toCompare:Object) : Boolean
```

Determines whether two points are equal. Two points are equal if they have the same *x* and *y* values.

Availability

Flash Lite 3.1

Parameters

toCompare : Object - The point to be compared.

Returns

Boolean - If the object is equal to this Point object, `true`; if it is not equal, `false`.

Example

The following example determines whether the values of one point are equal to the values of another point. If the objects are the same, `equals()` does not return the same result that the strict equality operator (`===`) does.

```
import flash.geom.Point;
var point_1:Point = new Point(1, 2);
var point_2:Point = new Point(1, 2);
var point_3:Point = new Point(4, 8);
trace(point_1.equals(point_2)); // true
trace(point_1.equals(point_3)); // false
trace(point_1 === point_2); // false
trace(point_1 === point_3); // false
```

interpolate (Point.interpolate method)

```
public static interpolate(pt1:Point, pt2:Point, f:Number) : Point
```

Determines a point between two specified points.

Availability

Flash Lite 3.1

Parameters

pt1 : Point - The first point.

pt2 : Point - The second point.

f : Number - The level of interpolation between the two points. Indicates where the new point will be, along the line between pt1 and pt2. If f=0, pt1 is returned; if f=1, pt2 is returned.

Returns

Point - The new, interpolated point.

Example

The following example locates the interpolated point (`interpolatedPoint`) half way (50%) between `point_1` and `point_2`.

```
import flash.geom.Point;
var point_1:Point = new Point(-100, -100);
var point_2:Point = new Point(50, 50);
var interpolatedPoint:Point = Point.interpolate(point_1, point_2, .5);
trace(interpolatedPoint.toString()); // (x=-25, y=-25)
```

length (Point.length property)

```
public length : Number
```

The length of the line segment from (0,0) to this point.

Availability

Flash Lite 3.1

Example

The following example creates a Point object, `myPoint`, and determines the length of a line from (0, 0) to that Point.

```
import flash.geom.Point;
var myPoint:Point = new Point(3,4);
trace(myPoint.length); // 5
```

See also

polar (Point.polar method)

normalize (Point.normalize method)

```
public normalize(length:Number) : Void
```

Scales the line segment between (0,0) and the current point to a set length.

Availability

Flash Lite 3.1

Parameters

length : Number - The scaling value. For example, if the current point is (0,5), and you normalize it to 1, the point returned is at (0,1).

Example

The following example extends the length of the normalizedPoint object from 5 to 10.

```
import flash.geom.Point;
var normalizedPoint:Point = new Point(3, 4);
trace(normalizedPoint.length); // 5
trace(normalizedPoint.toString()); // (x=3, y=4)
normalizedPoint.normalize(10);
trace(normalizedPoint.length); // 10
trace(normalizedPoint.toString()); // (x=6, y=8)
```

See also

length (Point.length property)

offset (Point.offset method)

```
public offset(dx:Number, dy:Number) : Void
```

Offsets the Point object by the specified amount. The value of `dx` is added to the original value of *x* to create the new *x* value. The value of `dy` is added to the original value of *y* to create the new *y* value.

Availability

Flash Lite 3.1

Parameters

dx : Number - The amount by which to offset the horizontal coordinate, *x*.

dy : Number - The amount by which to offset the vertical coordinate, *y*.

Example

The following example offsets a point's position by specified *x* and *y* amounts.

```
import flash.geom.Point;
var myPoint:Point = new Point(1, 2);
trace(myPoint.toString()); // (x=1, y=2)
myPoint.offset(4, 8);
trace(myPoint.toString()); // (x=5, y=10)
```

See also

add (Point.add method)

Point constructor

```
public Point(x:Number, y:Number)
```

Creates a new point. If you pass no parameters to this method, a point is created at (0,0).

Availability

Flash Lite 3.1

Parameters

x : Number - The horizontal coordinate. The default value is 0.

y : Number - The vertical coordinate. The default value is 0.

Example

The first example creates a Point object `point_1` with the default constructor.

```
import flash.geom.Point;
var point_1:Point = new Point();
trace(point_1.x); // 0
trace(point_1.y); // 0
```

The second example creates a Point object `point_2` with the coordinates $x = 1$ and $y = 2$.

```
import flash.geom.Point;
var point_2:Point = new Point(1, 2);
trace(point_2.x); // 1
trace(point_2.y); // 2
```

polar (Point.polar method)

```
public static polar(len:Number, angle:Number) : Point
```

Converts a pair of polar coordinates to a Cartesian point coordinate.

Availability

Flash Lite 3.1

Parameters

len : Number - The length coordinate of the polar pair.

angle : Number - The angle, in radians, of the polar pair.

Returns

Point - The Cartesian point.

Example

The following example creates a Point object `cartesianPoint` from the value of `angleInRadians` and a line length of 5. The `angleInRadians` value equal to Math.atan(3/4) is used because of the characteristics of right triangles with sides that have ratios of 3:4:5.

```
import flash.geom.Point;
var len:Number = 5;
var angleInRadians:Number = Math.atan(3/4);
var cartesianPoint:Point = Point.polar(len, angleInRadians);
trace(cartesianPoint.toString()); // (x=4, y=3)
```

When computers work with transcendental numbers, such as pi, some round-off error occurs because floating-point arithmetic has only finite precision. When you use `Math.PI`, consider using the `Math.round()` function, as shown in the following example.

```
import flash.geom.Point;
var len:Number = 10;
var angleInRadians:Number = Math.PI;
var cartesianPoint:Point = Point.polar(len, angleInRadians);
trace(cartesianPoint.toString()); // should be (x=-10, y=0), but is (x=-10,
y=1.22460635382238e-15)
trace(Math.round(cartesianPoint.y)); // 0
```

See also

length (Point.length property), round (Math.round method)

subtract (Point.subtract method)

```
public subtract(v:Point) : Point
```

Subtracts the coordinates of another point from the coordinates of this point to create a new point.

Availability

Flash Lite 3.1

Parameters

v: Point - The point to be subtracted.

Returns

Point - The new point.

Example

The following example creates `point_3` by subtracting `point_2` from `point_1`.

```
import flash.geom.Point;
var point_1:Point = new Point(4, 8);
var point_2:Point = new Point(1, 2);
var resultPoint:Point = point_1.subtract(point_2);
trace(resultPoint.toString()); // (x=3, y=6)
```

toString (Point.toString method)

```
public toString() : String
```

Returns a string that contains the values of the *x* and *y* coordinates. It has the form (*x*, *y*), so a point at 23,17 would report "(*x*=23, *y*=17)".

Availability
Flash Lite 3.1

Returns
String - A string.

Example
The following example creates a point and converts its values to a string in the format (x=x, y=y).

```
import flash.geom.Point;
var myPoint:Point = new Point(1, 2);
trace("myPoint: " + myPoint.toString()); // (x=1, y=2)
```

x (Point.x. property)

```
public x : Number
```

The horizontal coordinate of the point. The default value is 0.

Availability
Flash Lite 3.1

Example
The following example creates a Point object `myPoint` and sets the *x* coordinate value.

```
import flash.geom.Point;
var myPoint:Point = new Point();
trace(myPoint.x); // 0
myPoint.x = 5;
trace(myPoint.x); // 5
```

y (Point.y property)

```
public y : Number
```

The vertical coordinate of the point. The default value is 0.

Availability
Flash Lite 3.1

Example
The following example creates a Point object `myPoint` and sets the *y* coordinate value.

```
import flash.geom.Point;
var myPoint:Point = new Point();
trace(myPoint.y); // 0
myPoint.y = 5;
trace(myPoint.y); // 5
```

Rectangle (flash.geom.Rectangle)

```
Object
    |
    +-flash.geom.Rectangle

public class Rectangle
extends Object
```

The Rectangle class is used to create and modify Rectangle objects. A Rectangle object is an area defined by its position, as indicated by its top-left corner point (*x*, *y*), and by its width and its height. Be careful when you design these areas—if a rectangle is described as having its upper-left corner at 0,0 and has a height of 10 and a width of 20, the lower-right corner is at 9,19, because the count of width and height began at 0,0.

The `x`, `y`, `width`, and `height` properties of the Rectangle class are independent of each other; changing the value of one property has no effect on the others. However, the `right` and `bottom` properties are integrally related to those four—if you change `right`, you are changing `width`; if you change `bottom`, you are changing `height`, and so on. And you must have the `left` or *x* property established before you set `width` or `right` property.

Availability

Flash Lite 3.1

Property summary

Modifiers	Property	Description
	bottom : Number	The sum of the `y` and `height` properties.
	bottomright : Point	The location of the Rectangle object's bottom-right corner, determined by the values of the `x` and `y` properties.
	height : Number	The height of the rectangle in pixels.
	left : Number	The *x* coordinate of the top-left corner of the rectangle.
	right : Number	The sum of the `x` and `width` properties.
	size : Point	The size of the Rectangle object, expressed as a Point object with the values of the `width` and `height` properties.
	top : Number	The *y* coordinate of the top-left corner of the rectangle.
	topLeft : Point	The location of the Rectangle object's top-left corner determined by the *x* and *y* values of the point.
	width : Number	The width of the rectangle in pixels.
	x : Number	The *x* coordinate of the top-left corner of the rectangle.
	y : Number	The *y* coordinate of the top-left corner of the rectangle.

`Object constructor, __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)`

Constructor summary

Signature	Description
Rectangle (x : Number, y : Number, width : Number, height : Number)	Creates a new Rectangle object whose top-left corner is specified by the x and y parameters.

Method summary

Modifiers	Signature	Description
	clone () : Rectangle	Returns a new Rectangle object with the same values for the x, y, width, and height properties as the original Rectangle object.
	contains (x : Number, y : Number) : Boolean	Determines whether the specified point is contained within the rectangular region defined by this Rectangle object.
	containsPoint (pt : Point (flash.geom.Point)) : Boolean	Determines whether the specified point is contained within the rectangular region defined by this Rectangle object.
	containsRectangle (rect : Rectangle) : Boolean	Determines whether the Rectangle object specified by the rect parameter is contained within this Rectangle object.
	equals (toCompare : Object) : Boolean	Determines whether the object specified in the toCompare parameter is equal to this Rectangle object.
	inflate (dx : Number, dy : Number) : Void	Increases the size of the Rectangle object by the specified amounts.
	inflatePoint (pt : Point) : Void	Increases the size of the Rectangle object.
	intersection (toIntersect : Rectangle) : Rectangle	If the Rectangle object specified in the toIntersect parameter intersects with this Rectangle object, the intersection() method returns the area of intersection as a Rectangle object.
	intersects (toIntersect : Rectangle) : Boolean	Determines whether the object specified in the toIntersect parameter intersects with this Rectangle object.
	isEmpty () : Boolean	Determines whether or not this Rectangle object is empty.
	offset (dx : Number, dy : Number) : Void	Adjusts the location of the Rectangle object, as determined by its top-left corner, by the specified amounts.
	offsetPoint (pt : Point) : Void	Adjusts the location of the Rectangle object using a Point object as a parameter.
	setEmpty () : Void	Sets all of the Rectangle object's properties to 0.
	toString () : String	Builds and returns a string that lists the horizontal and vertical positions and the width and height of the Rectangle object.
	union (toUnion : Rectangle) : Rectangle	Adds two rectangles together to create a new Rectangle object, by filling in the horizontal and vertical space between the two rectangles.

"addProperty (Object.addProperty method)" on page 494, "hasOwnProperty (Object.hasOwnProperty method)" on page 497, isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

bottom (Rectangle.bottom property)

```
public bottom : Number
```

The sum of the `y` and `height` properties.

Availability

Flash Lite 3.1

Example

The following example creates a Rectangle object and changes the value of its `bottom` property from 15 to 30. Notice that the value of `rect.height` is also changed, from 10 to 25.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.height); // 10
trace(rect.bottom); // 15

rect.bottom = 30;
trace(rect.height); // 25
trace(rect.bottom); // 30
```

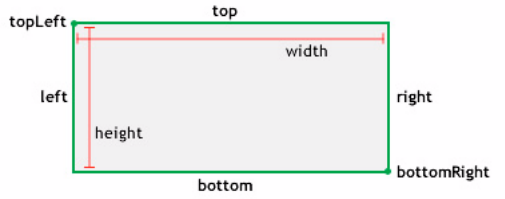
See also

`y (Rectangle.y property)`, `height (Rectangle.height property)`

bottomright (Rectangle.bottomright property)

```
public bottomRight : Point
```

The location of the Rectangle object's bottom-right corner, determined by the values of the `x` and `y` properties.



Availability

Flash Lite 3.1

Example

The following example sets the Rectangle object's `bottomRight` property using the values of the Point object. Notice that `rect.width` and `rect.height` are changed.

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.bottom); // 10
trace(rect.right); // 5
trace(rect.height); // 8
trace(rect.width); // 4

var myBottomRight:Point = new Point(16, 32);
rect.bottomRight = myBottomRight;
trace(rect.bottom); // 32
trace(rect.right); // 16
trace(rect.height); // 30
trace(rect.width); // 15
```

See also

Point (flash.geom.Point)

clone (Rectangle.clone method)

```
public clone() : Rectangle
```

Returns a new Rectangle object with the same values for the `x`, `y`, `width`, and `height` properties as the original Rectangle object.

Availability

Flash Lite 3.1

Returns

Rectangle - A new Rectangle object with the same values for the `x`, `y`, `width`, and `height` properties as the original Rectangle object.

Example

The following example creates three Rectangle objects and compares them. `rect_1` is created using the Rectangle constructor. `rect_2` is created by setting it equal to `rect_1`. And, `clonedRect` is created by cloning `rect_1`. Notice that while `rect_2` evaluates as being equal to `rect_1`, `clonedRect`, even though it contains the same values as `rect_1`, does not.

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(1, 2, 4, 8);
var rect_2:Rectangle = rect_1;
var clonedRect:Rectangle = rect_1.clone();

trace(rect_1 == rect_2); // true
trace(rect_1 == clonedFilter); // false

for(var i in rect_1) {
    trace(">> " + i + ": " + rect_1[i]);
    >> toString: [type Function]
    >> equals: [type Function]
    >> union: [type Function]
    >> intersects: [type Function]
    >> intersection: [type Function]
    >> containsRectangle: [type Function]
    >> containsPoint: [type Function]
    >> contains: [type Function]
    >> offsetPoint: [type Function]
    >> offset: [type Function]
    >> inflatePoint: [type Function]
    >> inflate: [type Function]
    >> size: (x=4, y=8)
    >> bottomRight: (x=5, y=10)
    >> topLeft: (x=1, y=2)
    >> bottom: 10
    >> top: 2
    >> right: 5
    >> left: 1
    >> isEmpty: [type Function]
    >> setEmpty: [type Function]
    >> clone: [type Function]
    >> height: 8
    >> width: 4
    >> y: 2
    >> x: 1
}

for(var i in clonedRect) {
    trace(">> " + i + ": " + clonedRect[i]);
    >> toString: [type Function]
    >> equals: [type Function]
    >> union: [type Function]
    >> intersects: [type Function]
    >> intersection: [type Function]
    >> containsRectangle: [type Function]
    >> containsPoint: [type Function]
```

```
    >> contains: [type Function]
    >> offsetPoint: [type Function]
    >> offset: [type Function]
    >> inflatePoint: [type Function]
    >> inflate: [type Function]
    >> size: (x=4, y=8)
    >> bottomRight: (x=5, y=10)
    >> topLeft: (x=1, y=2)
    >> bottom: 10
    >> top: 2
    >> right: 5
    >> left: 1
    >> isEmpty: [type Function]
    >> setEmpty: [type Function]
    >> clone: [type Function]
    >> height: 8
    >> width: 4
    >> y: 2
    >> x: 1
}
```

To further demonstrate the relationships between `rect_1`, `rect_2`, and `clonedRect`, the example below modifies the `x` property of `rect_1`. Modifying `x` demonstrates that the `clone()` method creates a new instance based on values of the `rect_1` instead of pointing to them in reference.

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(1, 2, 4, 8);
var rect_2:Rectangle = rect_1;
var clonedRect:Rectangle = rect_1.clone();

trace(rect_1.x); // 1
trace(rect_2.x); // 1
trace(clonedRect.x); // 1

rect_1.x = 10;

trace(rect_1.x); // 10
trace(rect_2.x); // 10
trace(clonedRect.x); // 1
```

See also

`x (Rectangle.x property)`, `y (Rectangle.y property)`, `width (Rectangle.width property)`, `height (Rectangle.height property)`

contains (Rectangle.contains method)

```
public contains(x:Number, y:Number) : Boolean
```

Determines whether the specified point is contained within the rectangular region defined by this Rectangle object.

Availability

Flash Lite 3.1

Parameters

x : Number - The *x*-value (horizontal position) of the point.

y : Number - The *y*-value (vertical position) of the point.

Returns

Boolean - If the specified point is contained in the Rectangle object, returns `true`; otherwise `false`.

Example

The following example creates a Rectangle object and tests whether each of three coordinate pairs falls within its boundaries.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(10, 10, 50, 50);
trace(rect.contains(59, 59)); // true
trace(rect.contains(10, 10)); // true
trace(rect.contains(60, 60)); // false
```

See also

`Point (flash.geom.Point)`

containsPoint (Rectangle.containsPoint method)

```
public containsPoint(pt:Point (flash.geom.Point)) : Boolean
```

Determines whether the specified point is contained within the rectangular region defined by this Rectangle object. This method is similar to the `Rectangle.contains()` method, except that it takes a Point object as a parameter.

Availability

Flash Lite 3.1

Parameters

pt : Point - The point, as represented by its *x*,*y* values.

Returns

Boolean - If the specified point is contained within this Rectangle object, returns `true`; otherwise `false`.

Example

The following example creates a Rectangle object and three Point objects, and tests whether each of the points falls within the boundaries of the rectangle.

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(10, 10, 50, 50);
trace(rect.containsPoint(new Point(10, 10))); // true
trace(rect.containsPoint(new Point(59, 59))); // true
trace(rect.containsPoint(new Point(60, 60))); // false
```

See also

`contains (Rectangle.contains method)`, `Point (flash.geom.Point)`

containsRectangle (Rectangle.containsRectangle method)

```
public containsRectangle(rect:Rectangle) : Boolean
```

Determines whether the Rectangle object specified by the `rect` parameter is contained within this Rectangle object. A Rectangle object is said to contain another if the second Rectangle object falls entirely within the boundaries of the first.

Availability

Flash Lite 3.1

Parameters

rect : Rectangle - The Rectangle object being checked.

Returns

Boolean - If the Rectangle object that you specify is contained by this Rectangle object, returns `true`; otherwise `false`.

Example

The following example creates four new Rectangle objects and determines whether rectangle A contains rectangle B, C, or D.

```
import flash.geom.Rectangle;

var rectA:Rectangle = new Rectangle(10, 10, 50, 50);
var rectB:Rectangle = new Rectangle(10, 10, 50, 50);
var rectC:Rectangle = new Rectangle(10, 10, 51, 51);
var rectD:Rectangle = new Rectangle(15, 15, 45, 45);

trace(rectA.containsRectangle(rectB)); // true
trace(rectA.containsRectangle(rectC)); // false
trace(rectA.containsRectangle(rectD)); // true
```

equals (Rectangle.equals method)

```
public equals(toCompare:Object) : Boolean
```

Determines whether the object specified in the `toCompare` parameter is equal to this Rectangle object. This method compares the `x`, `y`, `width`, and `height` properties of an object against the same properties of this Rectangle object.

Availability

Flash Lite 3.1

Parameters

toCompare : Object - The rectangle to compare to this Rectangle object.

Returns

Boolean - If the object has exactly the same values for the `x`, `y`, `width`, and `height` properties as this Rectangle object, returns `true`; otherwise `false`.

Example

In the following example, `rect_1` and `rect_2` are equal, but `rect_3` is not equal to the other two objects because its `x`, `y`, `width`, and `height` properties are not equal to those of `rect_1` and `rect_2`.

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(0, 0, 50, 100);
var rect_2:Rectangle = new Rectangle(0, 0, 50, 100);
var rect_3:Rectangle = new Rectangle(10, 10, 60, 110);

trace(rect_1.equals(rect_2)); // true;
trace(rect_1.equals(rect_3)); // false;
```

Even though the method signature expects only an abstract object, only other Rectangle instances are treated as equal.

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(0, 0, 50, 100);
var nonRect:Object = new Object();
nonRect.x = 0;
nonRect.y = 0;
nonRect.width = 50;
nonRect.height = 100;
trace(rect_1.equals(nonRect));
```

See also

`x (Rectangle.x property)`, `y (Rectangle.y property)`, `width (Rectangle.width property)`, `height (Rectangle.height property)`

height (Rectangle.height property)

```
public height : Number
```

The height of the rectangle in pixels. Changing the `height` value of a Rectangle object has no effect on the `x`, `y`, and `width` properties.

Availability

Flash Lite 3.1

Example

The following example creates a Rectangle object and changes its `height` property from 10 to 20. Notice that `rect.bottom` is also changed.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.height); // 10
trace(rect.bottom); // 15

rect.height = 20;
trace(rect.height); // 20
trace(rect.bottom); // 25
```

See also

x (Rectangle.x property), y (Rectangle.y property), width (Rectangle.width property)

inflate (Rectangle.inflate method)

```
public inflate(dx:Number, dy:Number) : Void
```

Increases the size of the Rectangle object by the specified amounts. The center point of the Rectangle object stays the same, and its size increases to the left and right by the `dx` value, and to the top and the bottom by the `dy` value.

Availability

Flash Lite 3.1

Parameters

dx : Number - The value to be added to the left and the right of the Rectangle object. The following equation is used to calculate the new width and position of the rectangle:

```
x -= dx;
width += 2 * dx;
```

dy : Number - The value to be added to the top and the bottom of the Rectangle object. The following equation is used to calculate the new height and position of the rectangle.

```
y -= dy;
height += 2 * dy;
```

Example

The following example creates a Rectangle object and increases the value of its `width` property by 16 * 2 (32) and of its `height` property by 32 * 2 (64)

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.toString()); // (x=1, y=2, w=4, h=8)

rect.inflate(16, 32);
trace(rect.toString()); // (x=-15, y=-30, w=36, h=72)
```

See also

x (Rectangle.x property), y (Rectangle.y property)

inflatePoint (Rectangle.inflatePoint method)

```
public inflatePoint(pt:Point) : Void
```

Increases the size of the Rectangle object. This method is similar to the `Rectangle.inflate()` method, except that it takes a Point object as a parameter.

The following two code examples give the same result:

```
rect1 = new flash.geom.Rectangle(0,0,2,5);
rect1.inflate(2,2)
rect1 = new flash.geom.Rectangle(0,0,2,5);
pt1 = new flash.geom.Point(2,2);
rect1.inflatePoint(pt1)
```

Availability

Flash Lite 3.1

Parameters

pt : Point - Increases the rectangle by the *x* and *y* coordinate values of the point.

Example

The following example creates a Rectangle object and inflates it by the *x* (horizontal) and *y* (vertical) amounts found in a point.

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(0, 0, 2, 5);
trace(rect.toString()); // (x=0, y=0, w=2, h=5

var myPoint:Point = new Point(2, 2);
rect.inflatePoint(myPoint);
trace(rect.toString()); // (x=-2, y=-2, w=6, h=9)
```

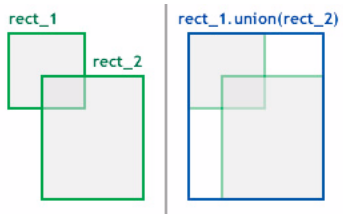
See also

Point (flash.geom.Point)

intersection (Rectangle.intersection method)

```
public intersection(toIntersect:Rectangle) : Rectangle
```

If the Rectangle object specified in the `toIntersect` parameter intersects with this Rectangle object, the `intersection()` method returns the area of intersection as a Rectangle object. If the rectangles do not intersect, this method returns an empty Rectangle object with its properties set to 0.



Availability

Flash Lite 3.1

Parameters

toIntersect : Rectangle - The Rectangle object to compare against to see if it intersects with this Rectangle object.

Returns

Rectangle - A Rectangle object that equals the area of intersection. If the rectangles do not intersect, this method returns an empty Rectangle object; that is, a rectangle with its `x`, `y`, `width`, and `height` properties set to 0.

Example

The following example determines the area where `rect_1` intersects `rect_2`.

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(0, 0, 50, 50);
var rect_2:Rectangle = new Rectangle(25, 25, 100, 100);
var intersectingArea:Rectangle = rect_1.intersection(rect_2);
trace(intersectingArea.toString()); // (x=25, y=25, w=25, h=25)
```

intersects (Rectangle.intersects method)

```
public intersects(toIntersect:Rectangle) : Boolean
```

Determines whether the object specified in the `toIntersect` parameter intersects with this Rectangle object. This method checks the `x`, `y`, `width`, and `height` properties of the specified Rectangle object to see if it intersects with this Rectangle object.

Availability

Flash Lite 3.1

Parameters

toIntersect : Rectangle - The Rectangle object to compare against this Rectangle object.

Returns

Boolean - If the specified object intersects with this Rectangle object, returns `true`; otherwise `false`.

Example

The following example determines whether `rectA` intersects with `rectB` or `rectC`.

```
import flash.geom.Rectangle;
var rectA:Rectangle = new Rectangle(10, 10, 50, 50);
var rectB:Rectangle = new Rectangle(59, 59, 50, 50);
var rectC:Rectangle = new Rectangle(60, 60, 50, 50);
var rectAIntersectsB:Boolean = rectA.intersects(rectB);
var rectAIntersectsC:Boolean = rectA.intersects(rectC);
trace(rectAIntersectsB); // true
trace(rectAIntersectsC); // false

var firstPixel:Rectangle = new Rectangle(0, 0, 1, 1);
var adjacentPixel:Rectangle = new Rectangle(1, 1, 1, 1);
var pixelsIntersect:Boolean = firstPixel.intersects(adjacentPixel);
trace(pixelsIntersect); // false
```

See also

`x (Rectangle.x property)`, `y (Rectangle.y property)`, `width (Rectangle.width property)`, `height (Rectangle.height property)`

isEmpty (Rectangle.isEmpty method)

```
public isEmpty() : Boolean
```

Determines whether or not this Rectangle object is empty.

Availability

Flash Lite 3.1

Returns

"`Boolean`" on page 218 - If the Rectangle object's width or height is less than or equal to 0, returns `true`; otherwise `false`.

Example

The following example creates an empty Rectangle object and verifies that it is empty.

```
import flash.geom.*;
var rect:Rectangle = new Rectangle(1, 2, 0, 0);
trace(rect.toString()); // (x=1, y=2, w=0, h=0)
trace(rect.isEmpty()); // true
```

The following example creates a non-empty Rectangle and makes it become empty.

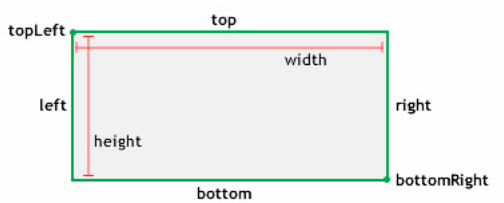
```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.isEmpty()); // false
rect.width = 0;
trace(rect.isEmpty()); // true
rect.width = 4;
trace(rect.isEmpty()); // false
rect.height = 0;
trace(rect.isEmpty()); // true
```

left (Rectangle.left property)

```
public left : Number
```

The *x* coordinate of the top-left corner of the rectangle. Changing the *x* value of a Rectangle object has no effect on the `y`, `width`, and `height` properties.

The `left` property is equal to the `x` property.



Availability
Flash Lite 3.1

Example
The following example changes the `left` property from 0 to 10. Notice that `rect.x` also changes.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle();
trace(rect.left); // 0
trace(rect.x); // 0

rect.left = 10;
trace(rect.left); // 10
trace(rect.x); // 10
```

See also
`x (Rectangle.x property)`, `y (Rectangle.y property)`, "`width (Rectangle.width property)`" on page 536, `height (Rectangle.height property)`

offset (Rectangle.offset method)

```
public offset(dx:Number, dy:Number) : Void
```

Adjusts the location of the Rectangle object, as determined by its top-left corner, by the specified amounts.

Availability
Flash Lite 3.1

Parameters
dx : Number - Moves the *x* value of the Rectangle object by this amount.

dy : Number - Moves the *y* value of the Rectangle object by this amount.

Example
The following example creates a Rectangle object and offsets its *x* and *y* values by 5 and 10, respectively.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.toString()); // (x=1, y=2, w=4, h=8)

rect.offset(16, 32);
trace(rect.toString()); // (x=17, y=34, w=4, h=8)
```

offsetPoint (Rectangle.offsetPoint method)

```
public offsetPoint(pt:Point) : Void
```

Adjusts the location of the Rectangle object using a Point object as a parameter. This method is similar to the `Rectangle.offset()` method, except that it takes a Point object as a parameter.

Availability
Flash Lite 3.1

Parameters
pt : Point - A Point object to use to offset this Rectangle object.

Example
The following example offsets a Rectangle by using the values found in a point.

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.toString()); // (x=1, y=2, w=4, h=8)

var myPoint:Point = new Point(16, 32);
rect.offsetPoint(myPoint);
trace(rect.toString()); // (x=17, y=34, w=4, h=8)
```

See also
Point (flash.geom.Point)

Rectangle constructor

```
public Rectangle(x:Number, y:Number, width:Number, height:Number)
```

Creates a new Rectangle object whose top-left corner is specified by the `x` and `y` parameters. If you call this constructor function without parameters, a rectangle with `x`, `y`, `width`, and `height` properties set to 0 is created.

Availability
Flash Lite 3.1

Parameters
x : Number - The *x* coordinate of the top-left corner of the rectangle.

y : Number - The *y* coordinate of the top-left corner of the rectangle.

width : Number - The width of the rectangle in pixels.

height : Number - The height of the rectangle in pixels.

Example
The following example creates a Rectangle object with the specified parameters.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 10, 50, 100);
trace(rect.toString()); // (x=5, y=10, w=50, h=100)
```

See also
x (Rectangle.x property), y (Rectangle.y property), "width (Rectangle.width property)" on page 536, height (Rectangle.height property)

right (Rectangle.right property)

```
public right : Number
```

The sum of the `x` and `width` properties.

Availability

Flash Lite 3.1

Example

The following example creates a Rectangle object and changes its `right` property from 15 to 30. Notice that `rect.width` also changes.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.width); // 10
trace(rect.right); // 15

rect.right = 30;
trace(rect.width); // 25
trace(rect.right); // 30
```

See also

`x (Rectangle.x property)`, `width (Rectangle.width property)`

setEmpty (Rectangle.setEmpty method)

```
public setEmpty() : Void
```

Sets all of the Rectangle object's properties to 0. A Rectangle object is empty if its width or height is less than or equal to 0.

This method sets the values of the `x`, `y`, `width`, and `height` properties to 0.

Availability

Flash Lite 3.1

Example

The following example creates a non-empty Rectangle object and makes it empty.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 10, 50, 100);
trace(rect.isEmpty()); // false
rect.setEmpty();
trace(rect.isEmpty()); // true
```

See also

`x (Rectangle.x property)`, `y (Rectangle.y property)`, `width (Rectangle.width property)`, `height (Rectangle.height property)`

size (Rectangle.size property)

```
public size : Point
```

The size of the Rectangle object, expressed as a Point object with the values of the `width` and `height` properties.

Availability

Flash Lite 3.1

Example

The following example creates a Rectangle object, retrieves its size (`size`), changes its size (`size`), and sets the new values on the Rectangle object. It is important to remember that the Point object used by the `size` property uses *x* and *y* values to represent the `width` and `height` properties of the Rectangle object.

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
var size:Point = rect.size;
trace(size.x); // 4;
trace(size.y); // 8;

size.x = 16;
size.y = 32;
rect.size = size;
trace(rect.x); // 1
trace(rect.y); // 2
trace(rect.width); // 16
trace(rect.height); // 32
```

See also

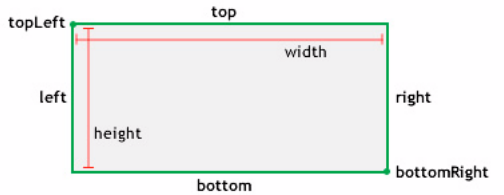
`Point (flash.geom.Point)`

top (Rectangle.top property)

```
public top : Number
```

The *y* coordinate of the top-left corner of the rectangle. Changing the value of the `top` property of a Rectangle object has no effect on the `x`, `width`, and `height` properties.

The value of the `top` property is equal to the value of the `y` property.



Availability

Flash Lite 3.1

Example

This example changes the value of the `top` property from 0 to 10. Notice that `rect.y` also changes.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle();
trace(rect.top); // 0
trace(rect.y); // 0

rect.top = 10;
trace(rect.top); // 10
trace(rect.y); // 10
```

See also

`x (Rectangle.x property)`, `y (Rectangle.y property)`, `width (Rectangle.width property)`, `height (Rectangle.height property)`

topLeft (Rectangle.topLeft property)

```
public topLeft : Point
```

The location of the Rectangle object's top-left corner determined by the *x* and *y* values of the point.

Availability

Flash Lite 3.1

Example

The following example sets the Rectangle object's `topLeft` property using the values in a Point object. Notice that `rect.x` and `rect.y` are changed.

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle();
trace(rect.left); // 0
trace(rect.top); // 0
trace(rect.x); // 0
trace(rect.y); // 0

var myTopLeft:Point = new Point(5, 15);
rect.topLeft = myTopLeft;
trace(rect.left); // 5
trace(rect.top); // 15
trace(rect.x); // 5
trace(rect.y); // 15
```

See also

Point (flash.geom.Point), x (Rectangle.x property), y (Rectangle.y property)

toString (Rectangle.toString method)

```
public toString() : String
```

Builds and returns a string that lists the horizontal and vertical positions and the width and height of the Rectangle object.

Availability

Flash Lite 3.1

Returns

String - A string that lists the value of each of the following properties of the Rectangle object: `x`, `y`, `width`, and `height`.

Example

The following example concatenates a string representation of `rect_1` with some helpful debugging text.

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(0, 0, 50, 100);
trace("Rectangle 1 : " + rect_1.toString()); // Rectangle 1 : (x=0, y=0, w=50, h=100)
```

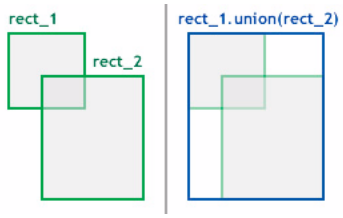
See also

x (Rectangle.x property), y (Rectangle.y property), width (Rectangle.width property), height (Rectangle.height property)

union (Rectangle.union method)

```
public union(toUnion:Rectangle) : Rectangle
```

Adds two rectangles together to create a new Rectangle object, by filling in the horizontal and vertical space between the two rectangles.



Availability
Flash Lite 3.1

Parameters
toUnion : Rectangle - A Rectangle object to add to this Rectangle object.

Returns
Rectangle - A new Rectangle object that is the union of the two rectangles.

Example
The following example creates a Rectangle object out of the union of two others.

For example, consider a rectangle with properties `x=20`, `y=50`, `width=60`, and `height=30` (20, 50, 60, 30), and a second rectangle with properties (150, 130, 50, 30). The union of these two rectangles is a larger rectangle that encompasses the two rectangles with the properties (20, 50, 180, 110).

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(20, 50, 60, 30);
var rect_2:Rectangle = new Rectangle(150, 130, 50, 30);
var combined:Rectangle = rect_1.union(rect_2);
trace(combined.toString()); // (x=20, y=50, w=180, h=110)
```

width (Rectangle.width property)

```
public width : Number
```

The width of the rectangle in pixels. Changing the value of the `width` property of a Rectangle object has no effect on the `x`, `y`, and `height` properties.

Availability
Flash Lite 3.1

Example

The following example creates a Rectangle object and changes its `width` property from 10 to 20. Notice that `rect.right` also changes.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.width); // 10
trace(rect.right); // 15

rect.width = 20;
trace(rect.width); // 20
trace(rect.right); // 25
```

See also

`x (Rectangle.x property)`, `y (Rectangle.y property)`, `height (Rectangle.height property)`

x (Rectangle.x property)

```
public x : Number
```

The *x* coordinate of the top-left corner of the rectangle. Changing the value of the *x* property of a Rectangle object has no effect on the `y`, `width`, and `height` properties.

The `x` property is equal to the `left` property.

Availability

Flash Lite 3.1

Example

The following example creates an empty Rectangle and sets its `x` property to 10. Notice that `rect.left` is also changed.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle();
trace(rect.x); // 0
trace(rect.left); // 0

rect.x = 10;
trace(rect.x); // 10
trace(rect.left); // 10
```

See also

`left (Rectangle.left property)`

y (Rectangle.y property)

```
public y : Number
```

The *y* coordinate of the top-left corner of the rectangle. Changing the value of the `y` property of a Rectangle object has no effect on the `x`, `width`, and `height` properties.

The `y` property is equal to the `top` property.

Availability

Flash Lite 3.1

Example

The following example creates an empty Rectangle and sets its `y` property to 10. Notice that `rect.top` is also changed.

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle();
trace(rect.y); // 0
trace(rect.top); // 0

rect.y = 10;
trace(rect.y); // 10
trace(rect.top); // 10
```

See also

x (Rectangle.x property), width (Rectangle.width property), height (Rectangle.height property) top (Rectangle.top property)

Security (System.security)

```
Object
    |
    +-System.security

public class security
extends Object
```

The System.security class contains methods that specify how SWF files in different domains can communicate with each other.

Availability

Flash Lite 2.0

Property summary

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property) prototype (Object.prototype property), __resolve (Object.__resolve property)

Method summary

Modifiers	Signature	Description
`static`	allowDomain(domain1:String) : Void	Lets SWF files and HTML files in the identified domains access objects and variables in the calling SWF file or in any other SWF file from the same domain as the calling SWF file.
`static`	allowInsecureDomain(domain:String) : Void	Lets SWF files and HTML files in the identified domains access objects and variables in the calling SWF file, which is hosted using the HTTPS protocol.
`static`	loadPolicyFile(url:String) : Void	Loads a cross-domain policy file from a location specified by the `url` parameter.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

allowDomain (security.allowDomain method)

```
public static allowDomain(domain1:String) : Void
```

Lets SWF files and HTML files in the identified domains access objects and variables in the calling SWF file or in any other SWF file from the same domain as the calling SWF file.

In files playing in Flash Player 7 or later, the parameters passed must follow exact-domain naming rules. For example, to allow access by SWF files hosted at either www.domain.com or store.domain.com, both domain names must be passed:

```
// For Flash Player 6
System.security.allowDomain("domain.com");
// Corresponding commands to allow access by SWF files
// that are running in Flash Player 7 or later
System.security.allowDomain("www.domain.com", "store.domain.com");
```

Also, for files running in Flash Player 7 or later, you can't use this method to let SWF files hosted using a secure protocol (HTTPS) allow access from SWF files hosted in nonsecure protocols; you must use `System.security.allowInsecureDomain()` instead.

Occasionally, you might encounter the following situation: You load a child SWF file from a different domain and want to allow the child SWF file to script the parent SWF file, but you don't know the final domain from which the child SWF file will originate. This can happen, for example, when you use load-balancing redirects or third-party servers.

In this situation, you can use the `MovieClip._url` property as an argument to this method. For example, if you load a SWF file into `my_mc`, you can call `System.security.allowDomain(my_mc._url)`.

If you do this, be sure to wait until the SWF file in `my_mc` is loaded, because the `_url` property does not have its final, correct value until the file is completely loaded. The best way to determine when a child SWF finishes loading is to use `MovieClipLoader.onLoadComplete`.

The opposite situation can also occur; that is, you might create a child SWF file that wants to allow its parent to script it, but doesn't know what the domain of its parent will be. In this situation, call `System.security.allowDomain(_parent._url)` from the child SWF. In this situation, you don't have to wait for the parent SWF file to load; the parent is already loaded by the time the child loads.

Availability

Flash Lite 2.0

Parameters

domain1 : String - One or more strings that specify domains that can access objects and variables in the SWF file that contains the `System.Security.allowDomain()` call. The domains can be formatted in the following ways:

- `"domain.com"`
- `"http://domain.com"`
- `"http://IPaddress"`

Example

The SWF file located at www.macromedia.com/MovieA.swf contains the following lines:

```
System.security.allowDomain("www.shockwave.com");
loadMovie("http://www.shockwave.com/MovieB.swf", my_mc);
```

Because MovieA contains the `allowDomain()` call, MovieB can access the objects and variables in MovieA. If MovieA didn't contain this call, the Flash security implementation would prevent MovieB from accessing MovieA's objects and variables.

See also

`onLoadComplete (MovieClipLoader.onLoadComplete event listener)`, `_parent (MovieClip._parent property)`, `_url (MovieClip._url property)`, `allowInsecureDomain (security.allowInsecureDomain method)`

allowInsecureDomain (security.allowInsecureDomain method)

```
public static allowInsecureDomain(domain:String) : Void
```

Lets SWF files and HTML files in the identified domains access objects and variables in the calling SWF file, which is hosted using the HTTPS protocol. It also lets the SWF files in the identified domains access any other SWF files in the same domain as the calling SWF file.

By default, SWF files hosted using the HTTPS protocol can be accessed only by other SWF files hosted using the HTTPS protocol. This implementation maintains the integrity provided by the HTTPS protocol.

Adobe does not recommend using this method to override the default behavior because it compromises HTTPS security. However, you might need to do so, for example, if you must permit access to HTTPS files published for Flash Player 7 or later from HTTP files published for Flash Player 6.

A SWF file published for Flash Player 6 can use `System.security.allowDomain()` to permit HTTP to HTTPS access. However, because security is implemented differently in Flash Player 7, you must use `System.Security.allowInsecureDomain()` to permit such access in SWF files published for Flash Player 7 or later.

Note: It is sometimes necessary to call `System.security.allowInsecureDomain()` with an argument that exactly matches the domain of the SWF file in which this call appears. This is different from `System.security.allowDomain()`, which is never necessary to call with a SWF file's own domain as an argument. The reason this is sometimes necessary with `System.security.allowInsecureDomain()` is that, by default, a SWF file at http://foo.com is not allowed to script a SWF file at https://foo.com, even though the domains are identical.

Availability
Flash Lite 2.0

Parameters
domain : String - An exact domain name, such as www.myDomainName.com or store.myDomainName.com.

Example
In the following example, you host a math test on a secure domain so that only registered students can access it. You have also developed a number of SWF files that illustrate certain concepts, which you host on an insecure domain. You want students to access the test from the SWF file that contains information about a concept.

```
// This SWF file is at https://myEducationSite.somewhere.com/mathTest.swf
// Concept files are at http://myEducationSite.somewhere.com
System.security.allowInsecureDomain("myEducationSite.somewhere.com");
```

See also
allowDomain (security.allowDomain method)

loadPolicyFile (security.loadPolicyFile method)

```
public static loadPolicyFile(url:String) : Void
```

Loads a cross-domain policy file from a location specified by the `url` parameter. Flash Lite player uses policy files as a permission mechanism to permit Flash movies to load data from servers other than their own.

Flash Player 7.0.14.0 looked for policy files in only one location: /crossdomain.xml on the server to which a data-loading request was being made. For an XMLSocket connection attempt, Flash Player 7.0.14.0 looked for /crossdomain.xml on an HTTP server on port 80 in the subdomain to which the XMLSocket connection attempt was being made. Flash Player 7.0.14.0 (and all earlier players) also restricted XMLSocket connections to ports 1024 and later.

With the addition of `System.security.loadPolicyFile()`, Flash Player 7.0.19.0 can load policy files from arbitrary locations, as shown in the following example:

```
System.security.loadPolicyFile("http://foo.com/sub/dir/pf.xml");
```

This causes Flash player to retrieve a policy file from the specified URL. Any permissions granted by the policy file at that location will apply to all content at the same level or lower in the virtual directory hierarchy of the server. The following code continues the previous example:

```
loadVariables("http://foo.com/sub/dir/vars.txt") // allowed
loadVariables("http://foo.com/sub/dir/deep/vars2.txt") // allowed
loadVariables("http://foo.com/elsewhere/vars3.txt") // not allowed
```

You can use `loadPolicyFile()` to load any number of policy files. When considering a request that requires a policy file, Flash player always waits for the completion of any policy file downloads before denying a request. As a final fallback, if no policy file specified with `loadPolicyFile()` authorizes a request, Flash player consults the original default location, /crossdomain.xml.

Using the `xmlsocket` protocol along with a specific port number, lets you retrieve policy files directly from an XMLSocket server, as shown in the following example:

```
System.security.loadPolicyFile("xmlsocket://foo.com:414");
```

This causes Flash player to attempt to retrieve a policy file from the specified host and port. Any port can be used, not only ports 1024 and higher. Upon establishing a connection with the specified port, Flash player transmits `<policy-file-request />`, terminated by a `null` byte. An XMLSocket server can be configured to serve both policy files and normal XMLSocket connections over the same port, in which case the server should wait for `<policy-file-request />` before transmitting a policy file. A server can also be set up to serve policy files over a separate port from standard connections, in which case it can send a policy file as soon as a connection is established on the dedicated policy file port. The server must send a null byte to terminate a policy file, and may thereafter close the connection; if the server does not close the connection, Flash player does so upon receiving the terminating `null` byte.

A policy file served by an XMLSocket server has the same syntax as any other policy file, except that it must also specify the ports to which access is granted. When a policy file comes from a port lower than 1024, it can grant access to any ports; when a policy file comes from port 1024 or higher, it can grant access only to other ports 1024 and higher. The allowed ports are specified in a `"to-ports"` attribute in the `<allow-access-from>` tag. Single port numbers, port ranges, and wildcards are all allowed. The following example shows an XMLSocket policy file:

```
<cross-domain-policy>
<allow-access-from domain="*" to-ports="507" />
<allow-access-from domain="*.foo.com" to-ports="507,516" />
<allow-access-from domain="*.bar.com" to-ports="516-523" />
<allow-access-from domain="www.foo.com" to-ports="507,516-523" />
<allow-access-from domain="www.bar.com" to-ports="*" />
</cross-domain-policy>
```

A policy file obtained from the old default location—/crossdomain.xml on an HTTP server on port 80—implicitly authorizes access to all ports 1024 and above. There is no way to retrieve a policy file to authorize XMLSocket operations from any other location on an HTTP server; any custom locations for XMLSocket policy files must be on an XMLSocket server.

Because the ability to connect to ports lower than 1024 is new, a policy file loaded with `loadPolicyFile()` must always authorize this connection, even when a movie clip is connecting to its own subdomain.

Availability
Flash Lite 2.0

Parameters
url : String - A string; the URL where the cross-domain policy file to be loaded is located.

Selection

```
Object
    |
    +-Selection

public class Selection
extends Object
```

The Selection class lets you set and control the text field in which the insertion point is located (that is, the field that has focus). Selection-span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

There is no constructor function for the Selection class, because there can be only one currently focused field at a time.

The Selection object is valid only when a device supports inline text entry. If a device does not support inline text entry, and instead relies on an FEP (front-end processor) to enter text, all calls to the Selection object are ignored.

Availability

Flash Lite 2.0

Property summary

Properties inherited from class Object

`constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)`

Event summary

Event	Description
onSetFocus = `function([oldfocus], [newfocus]) {}`	Notified when the input focus changes.

Method summary

Modifiers	Signature	Description
`static`	addListener(`listener:` Object) : Void	Registers an object to receive keyboard focus change notifications.
`static`	getFocus() : String	Returns a string specifying the target path of the object that has focus.
`static`	removeListener(`listener:`Object) : Boolean	Removes an object previously registered with the `Selection.addListener()` method.
`static`	setFocus(`newFocus:`Object) : Boolean	Gives focus to the selectable (editable) text field, button, or movie clip, that the `newFocus` parameter specifies.
`static`	setSelection(`beginIndex:`Number, `endIndex:`Number) : Void	Sets the selection span of the currently focused text field.

Methods inherited from class Object

`addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)`

addListener (Selection.addListener method)

```
public static addListener(listener:Object) : Void
```

Registers an object to receive keyboard focus change notifications. When the focus changes (for example, whenever the `Selection.setFocus()` method is invoked), all listening objects registered with `addListener()` have their `onSetFocus()` method invoked. Multiple objects can listen for focus change notifications. If the specified listener is already registered, no change occurs.

Availability
Flash Lite 2.0

Parameters
listener : Object - A new object with an `onSetFocus` method.

Example
In the following example, you create two input text fields at runtime, setting the borders for each text field to `true`. This code creates a new (generic) ActionScript object named `focusListener`. This object defines for itself an `onSetFocus` property, to which it assigns a function. The function takes two parameters: a reference to the text field that lost focus, and one to the text field that gained focus. The function sets the `border` property of the text field that lost focus to `false`, and sets the border property of the text field that gained focus to `true`:

```
this.createTextField("one_txt", 99, 10, 10, 200, 20);
this.createTextField("two_txt", 100, 10, 50, 200, 20);
one_txt.border = true;
one_txt.type = "input";
two_txt.border = true;
two_txt.type = "input";

var focusListener:Object = new Object();
focusListener.onSetFocus = function(oldFocus_txt, newFocus_txt) {
    oldFocus_txt.border = false;
    newFocus_txt.border = true;
};
Selection.addListener(focusListener);
```

See also
setFocus (Selection.setFocus method)

getFocus (Selection.getFocus method)

```
public static getFocus() : String
```

Returns a string specifying the target path of the object that has focus.

- If a TextField object has focus, and the object has an instance name, the `getFocus()` method returns the target path of the TextField object. Otherwise, it returns the TextField variable name.
- If a Button object or button movie clip has focus, the `getFocus()` method returns the target path of the Button object or button movie clip.
- If neither a TextField object, Button object, Component instance, nor button movie clip has focus, the `getFocus()` method returns `null`.

Availability
Flash Lite 2.0

Returns

String - A string or `null`.

Example

The following example creates a text field to output the path of the currently focused object. It then uses an interval function to periodically update the field. To test this, add several button instances to the stage with different instance names, and then add the following ActionScript to your AS or FLA file.

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 150, 25);

function FocusUpdate()
{
    s = Selection.getFocus();
    if ( s )
    {
        status_txt.text = s;
    }
}

setInterval( FocusUpdate, 100 );
```

See also

`onSetFocus (Selection.onSetFocus event listener)`, `setFocus (Selection.setFocus method)`

onSetFocus (Selection.onSetFocus event listener)

```
onSetFocus = function([oldfocus], [newfocus]) {}
```

Notified when the input focus changes. To use this listener, you must create a listener object. You can then define a function for this listener and use the `Selection.addListener()` method to register the listener with the Selection object, as in the following code:

```
var someListener:Object = new Object();
someListener.onSetFocus = function () {
    // statements
}
Selection.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

Availability

Flash Lite 2.0

Parameters

oldfocus: [optional] - The object losing focus.

newfocus: [optional] - The object receiving focus.

Example

The following example demonstrates how to determine when input focus changes in a SWF file between several dynamically created text fields. Enter the following ActionScript into a FLA or AS file and then test the document:

```
this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {
    if (this[i] instanceof TextField) {
    this[i].border = true;
    this[i].type = "input";
    }
}

this.createTextField("status_txt", this.getNextHighestDepth(), 200, 10, 300, 100);
status_txt.html = true;
status_txt.multiline = true;

var someListener:Object = new Object();
someListener.onSetFocus = function(oldFocus, newFocus) {
    status_txt.htmlText = "<b>setFocus triggered</b>";
    status_txt.htmlText += "<textformat tabStops='[20,80]'>";
    status_txt.htmlText += " \toldFocus:\t"+oldFocus;
    status_txt.htmlText += " \tnewFocus:\t"+newFocus;
    status_txt.htmlText += " \tgetFocus:\t"+Selection.getFocus();
    status_txt.htmlText += "</textformat>";
};
Selection.addListener(someListener);
```

See also

addListener (Selection.addListener method), setFocus (Selection.setFocus method)

removeListener (Selection.removeListener method)

```
public static removeListener(listener:Object) : Boolean
```

Removes an object previously registered with the `Selection.addListener()` method.

Availability

Flash Lite 2.0

Parameters

listener : Object - The object that no longer receives focus notifications.

Returns

Boolean - If the listener object was successfully removed, the method returns a `true` value. If the listener object was not successfully removed—for example, if `listener` was not on the Selection object's listener list—the method returns a value of `false`.

Example

The following ActionScript dynamically creates several text field instances. When you select a text field, information appears in the Output panel. When you click the `remove_btn` instance, the listener is removed and information no longer appears in the Output panel.

```
this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {
    if (this[i] instanceof TextField) {
    this[i].border = true;
    this[i].type = "input";
    }
}

var selectionListener:Object = new Object();
selectionListener.onSetFocus = function(oldFocus, newFocus) {
    trace("Focus shifted from "+oldFocus+" to "+newFocus);
};
Selection.addListener(selectionListener);

remove_btn.onRelease = function() {
    trace("removeListener invoked");
    Selection.removeListener(selectionListener);
};
```

See also

addListener (Selection.addListener method)

setFocus (Selection.setFocus method)

```
public static setFocus(newFocus:Object) : Boolean
```

Gives focus to the selectable (editable) text field, button, or movie clip, that the `newFocus` parameter specifies. You can use dot or slash notation to specify the path. You can also use a relative or absolute path. If you are using ActionScript 2.0, you must use dot notation.

If `null` is passed, the current focus is removed.

Availability

Flash Lite 2.0

Parameters

newFocus : Object - An object such as a button, movie clip, or text field instance, or a string specifying the path to a button, movie clip, or text field instance.

Returns

Boolean - A Boolean value; `true` if the focus attempt is successful, `false` if it fails.

Example

In the following example, the text field focuses on the `username_txt` text field when it is running in a browser window. If the user does not fill in one of the required text fields (`username_txt` and `password_txt`), the cursor automatically focuses in the text field that is missing data. For example, if the user does not type anything into the `username_txt` text field and clicks the submit button, an error message appears and the cursor focuses in the `username_txt` text field.

```
this.createTextField("status_txt", this.getNextHighestDepth(), 100, 70, 100, 22);
this.createTextField("username_txt", this.getNextHighestDepth(), 100, 100, 100, 22);
this.createTextField("password_txt", this.getNextHighestDepth(), 100, 130, 100, 22);
this.createEmptyMovieClip("submit_mc", this.getNextHighestDepth());
submit_mc.createTextField("submit_txt", this.getNextHighestDepth(), 100, 160, 100, 22);
submit_mc.submit_txt.autoSize = "center";
submit_mc.submit_txt.text = "Submit";
submit_mc.submit_txt.border = true;
submit_mc.onRelease = checkForm;
username_txt.border = true;
password_txt.border = true;
username_txt.type = "input";
password_txt.type = "input";
password_txt.password = true;
Selection.setFocus("username_txt");
fscommand("activateTextField");
//
function checkForm():Boolean {
    if (username_txt.text.length == 0) {
    status_txt.text = "fill in username";
    Selection.setFocus("username_txt");
    fscommand("activateTextField");
    return false;
    }
    if (password_txt.text.length == 0) {
    status_txt.text = "fill in password";
    Selection.setFocus("password_txt");
    fscommand("activateTextField");
    return false;
    }
    status_txt.text = "success!";
    Selection.setFocus(null);
    return true;
}
```

See also

getFocus (Selection.getFocus method)

setSelection (Selection.setSelection method)

```
public static setSelection(beginIndex:Number, endIndex:Number) : Void
```

Sets the selection span of the currently focused text field. The new selection span begins at the index specified in the `beginIndex` parameter, and ends at the index specified in the `endIndex` parameter. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on). This method has no effect if no text field currently has focus. When you call the `setSelection()` method and a text control has focus, the selection highlight is drawn only when the text field is being actively edited. The `setSelection()` method can be invoked after `Selection.setFocus()` or from within an `onSetFocus()` event handler, but any selection is visible only following a call to the `fscommand activateTextField` command.

Availability

Flash Lite 2.0

Parameters

beginIndex : Number - The beginning index of the selection span.

endIndex : Number - The ending index of the selection span.

Example

The following ActionScript code creates a text field at runtime and adds a string to it. Then it assigns an event handler for the `onSetFocus` event that selects all the text in the text field and activates the editing session.

Note: If the `Selection.setSelection()` method is called, the text is not drawn on screen until the text field is activated (following a call to the `fscommand activateTextField` command).

```
this.createTextField("myText_txt", 99, 10, 10, 200, 30);
myText_txt.type = "input";
myText_txt.text = "this is my text";
myText_txt.onSetFocus = function(){
    Selection.setSelection(0,myText_txt.text.length);
    fscommand("activateTextField");
}
```

The following example illustrates how the `endIndex` parameter is not inclusive. In order to select the first character, you must use an `endIndex` of 1, not 0. If you change the `endIndex` parameter to 0, nothing will be selected.

```
this.createTextField("myText_txt", 99, 10, 10, 200, 30);
myText_txt.text = "this is my text";
this.onEnterFrame = function () {
    Selection.setFocus("myText_txt");
    Selection.setSelection(0, 1);
    delete this.onEnterFrame;
}
```

SharedObject

```
Object
    |
    +-SharedObject
```

```
public dynamic class SharedObject
extends Object
```

The Flash Lite version of the SharedObject class allows Flash SWF files to save data to the device when it is closed and load that data from the device when it is played again. Flash Lite shared objects store a set of name-value pairs to the device.

Note: The name "SharedObject" is derived from the Flash SharedObject class. The Flash version of this class allows multiple Flash SWF files to share their saved data. However, versions of Flash Lite earlier than Flash Lite 3.1.5 do not support sharing data between different SWF files. Flash Lite 3.1.5 does allow multiple SWF files to share their saved data.

In Flash Lite, a SWF file is considered to be a different version if it was modified from the original version, even if it has the same name. This is different than in Flash player, where a SWF file is considered to be the same if its URL and name are the same, even if the SWF file was modified.

To maintain consistency with the Flash platform, the same ActionScript construct and calling conventions are used for the Flash Lite player.

The following examples describe the potential of using shared objects:

- A Flash application can be used as a user interface for a service that enables the user to search used car listings. The application connects to a server that provides listings of cars based on the search terms and preferences that the user enters. The Flash application can save the last search the user made and prefill the forms the next time the SWF file is played. To do this, you create a SharedObject instance that stores search parameters each time the user makes a new search. When the SWF file closes, the player saves the data in the shared object to the device. The next time the SWF file plays, the Flash Lite player loads the shared object and prefills the search form with the same search data the user entered the previous time.
- A Flash application can be used as a user interface for a service that allows users to search for music reviews. The application lets users store information about their favorite albums. The information can be stored on the remote server, but this causes problems if the application cannot connect to the service. Also, retrieving the data from a remote service can be slow and detract from the user experience. Shared objects enable the application to store information about the albums to the device and load it quickly when needed.

Note: Because space is limited on mobile devices, the data is not completely persistent; in some situations, the platform could delete the oldest data from the device.

To create a local shared object, use the following syntax:

```
var so:shared object = shared object.getLocal("mySharedObject");
```

Reading and writing data on a handset can be slow. To ensure that data is immediately available when the application requests it from the device, Flash Lite 2.0 requires you to set up a listener. The player invokes the listener when the device has loaded the shared object's data. Methods that access the SharedObject instance returned by the call to `getLocal()` should wait until the listener is invoked before attempting any operations.

Availability

Flash Lite 2.0

Example

In the following example, a SWF file creates a listener function named `Prefs` and then creates a shared object. The player calls the `loadCompletePrefs` function when the data is available.

```
function loadCompletePrefs (mySO:SharedObject) {
    if (0 == mySO.getSize() )
    {
    // If the size is 0, we need to initialize the data:
        mySO.data.name = "Sigismund";
        mySO.data.email = "siggy@macromedia.com";
    }
    else
    {
        // Trace all the data in mySO:
        trace( "Prefs:" );
        for (var idx in mySO.data) {
        trace( " " + idx +": " + mySO.data[idx] );
        }
    }
}

SharedObject.addListener( "Prefs", loadCompletePrefs );

// We can now create the shared object:
var Prefs:SharedObject = SharedObject.getLocal("Prefs");
```

When the player has notified the listener that the data is available, the application can use the shared object returned from the call to the `getLocal()` method in the same way a shared object is used in Flash. The application can add, modify, or remove properties while the content is playing. When the content is unloaded, the shared object might be written to the device; however, to guarantee that the shared object will be written to the device, the application must force a write operation by calling the `flush()` method.

Flash Lite shared objects are available only to locally stored SWF files. SWF files playing back in a network-enabled browser cannot use Flash Lite shared objects.

The total amount of storage for Flash Lite shared objects per SWF file is limited by the device to a predetermined size. You can determine this size by using the `SharedObject.getMaxSize()` method.

Note: Remote shared objects are not supported in Flash Lite 2.0.

See also

`flush (SharedObject.flush method)`, `onStatus (SharedObject.onStatus handler)`

Property summary

Modifiers	Property	Description
	data : Object	The collection of attributes assigned to the `data` property of the object.

Properties inherited from class Object

`constructor (Object.constructor property)`, `__proto__ (Object.__proto__ property)`, `prototype (Object.prototype property)`, `__resolve (Object.__resolve property)`

Event summary

Event	Description
onStatus = `function(infoObject:Object) {}`	Invoked every time an error, warning, or informational note is posted for a shared object.

Method summary

Modifiers	Signature	Description
`static`	addListener(`objectName:String, notifyFunction:Function) : Void`	Creates an event listener that the Flash Lite player invokes when the player has loaded the shared object data from the device.
	clear() : Void	Purges all the data from the shared object and deletes the shared object from the disk.
	flush(`minDiskSpace:Number`) : Object	Writes shared object to a local, persistent file.
`static`	getLocal(`name:String`) : SharedObject	Returns a reference to a locally persistent shared object that is available only to the current client.

Modifiers	Signature	Description
static	getMaxSize() : Number	Returns the total number of bytes the SWF file can use to store mobile shared objects on the device.
	getSize() : Number	Gets the current size of the shared object, in bytes.
static	removeListener(objectName:String)	Removes any listeners that were added using the addListener() method.

Methods inherited from class Object

addProperty (Object.addProperty method),hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method)isPrototypeOf (Object.isPrototypeOf method),registerClass (Object.registerClass method),toString (Object.toString method)unwatch (Object.unwatch method),valueOf (Object.valueOf method),watch (Object.watch method)

addListener (SharedObject.addListener method)

```
public static addListener(objectName:String, notifyFunction:Function) : Void
```

Creates an event listener that the Flash Lite player invokes when the player has loaded the shared object data from the device. Methods that access the SharedObject instance that the call returns to the `getLocal()` method should wait until this function is invoked before attempting any operations.

Availability

Flash Lite 2.0

Parameters

objectName : String - A string that represents the name of the shared object.

notifyFunction : Function - The name of a function the player calls to notify the application that the `getLocal()` method has executed and the data is finished loading.

clear (SharedObject.clear method)

```
public clear() : Void
```

Purges all the data from the shared object and deletes the shared object from the disk. The reference to `my_so` is still active, and `my_so` is now empty.

Availability

Flash Lite 2.0

Example

The following example sets data in the shared object, and then empties all of the data from the shared object:

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.name = "Hector";
trace("before my_so.clear():");
for (var prop in my_so.data) {
    trace("\t"+prop);
}
trace("");
my_so.clear();
trace("after my_so.clear():");
for (var prop in my_so.data) {
    trace("\t"+prop);
}
```

This ActionScript displays the following message in the Output panel:

```
before my_so.clear():
    name

after my_so.clear():
```

data (SharedObject.data property)

```
public data : Object
```

The collection of attributes assigned to the `data` property of the object. Each attribute can be an object of any basic ActionScript or JavaScript type—Array, Number, Boolean, and so on. For example, the following lines assign values to various aspects of a shared object.

Note: For Flash Lite, if the shared object listener has not been invoked, the `data` property could contain undefined values. For details, see the description of the `addListener()` method.

```
var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

for (var prop in my_so.data) {
    trace(prop+": "+my_so.data[prop]);
}

soResult = "";
for (var prop in my_so.data) {
    soResult += prop+": "+my_so.data[prop] +"\n";
}
result.text = soResult;
```

All attributes of a shared object's `data` property are saved if the object is persistent and the shared object contains the following information:

```
userName: Ramona
adminPrivileges: true
itemNumbers: 101,346,483
```

Note: Do not assign values directly to the `data` property of a shared object (for example, `so.data = someValue`). Flash ignores these assignments.

To delete attributes for local shared objects, use code such as `delete so.data.attributeName`; setting an attribute to `null` or `undefined` for a local shared object does not delete the attribute.

To create *private* values for a shared object—values that are available only to the client instance while the object is in use and are not stored with the object when it is closed—create properties that are not named `data` to store them, as shown in the following example:

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.favoriteColor = "blue";
my_so.favoriteNightClub = "The Bluenote Tavern";
my_so.favoriteSong = "My World is Blue";

for (var prop in my_so) {
    trace(prop+": "+my_so[prop]);
}
```

The shared object contains the following data:

```
favoriteSong: My World is Blue
favoriteNightClub: The Bluenote Tavern
favoriteColor: blue
data: [object Object]
```

Availability

Flash Lite 2.0

Example

The following example saves text to a shared object named `my_so` (for the complete example, see `SharedObject.getLocal()`):

```
var my_so:SharedObject = SharedObject.getLocal("savedText");

// myText is an input text field and inputText is a dynamic text field.
myText.text = my_so.data.myTextSaved;
// Assign an empty string to myText_ti if the shared object is undefined
// to prevent the text input box from displaying "undefined" when
// this script is first run.
if (myText.text == "undefined") {
    myText.text = "";
}

changedListener = new Object();
changedListener.onChanged = function (changedField) {
    my_so.data.myTextSaved = changedField.text;
    inputText.text = "";
    inputText.text = my_so.data.myTextSaved;
}
myText.addListener(changedListener);
```

flush (SharedObject.flush method)

```
public flush(minDiskSpace:Number) : Object
```

Writes shared object to a local, persistent file. To guarantee that the shared object will be written to the device, the application must force a write operation by calling the `flush()` method.

Unlike in Flash Player, the write operation is asynchronous and the result is not immediately available.

Availability
Flash Lite 2.0

Parameters
minDiskSpace : Number - An integer specifying the number of bytes that must be allotted for this object. The default value is 0.

Returns
Object - A Boolean value, `true` or `false`; or a string value of `"pending"`. The `flush()` method returns `pending` for most requests, with the following exceptions:

- If there is no need to write data (that is, the data has already been written), `flush()` returns `true`.
- If the `minimumDiskSpace` parameter exceeds the maximum space available for a SWF file, or the remaining space available for a SWF file, or if there was an error processing the request, `flush()` returns `false`.

If the `flush()` method returns `pending`, the Flash Lite player can show a dialog box asking the user to free up space to increase the amount of disk space available to shared objects. To allow space for the shared object to expand when it is saved in the future, which avoids return values of `pending` , you can pass a value for `minimumDiskSpace`. When the Flash Lite player tries to write the file, it searches for the number of bytes passed to `minimumDiskSpace`, instead of searching for enough space to save the shared object at its current size.

Example
The following example handles the possible return values for the `flush()` method:

```
so_big = SharedObject.getLocal("large");

so_big.data.name = "This is a long string of text.";
so_big.flush();
var flushResult = so_big.flush();

switch (flushResult) {
case 'pending' :
    result.text += "pending";
    break;
case true :
    result.text += "Data was flushed.";
    break;
case false :
    result.text += "Test failed. Data was not flushed.";
    break;
}
```

See also
clear (SharedObject.clear method), onStatus (SharedObject.onStatus handler)

getLocal (SharedObject.getLocal method)

```
public static getLocal(name:String) : SharedObject
```

Returns a reference to a locally persistent shared object that is available only to the current client. If the shared object does not already exist, `getLocal()` creates one. This method is a static method of the SharedObject class.

Note: In Flash Lite versions earlier than 3.1.5, a shared object cannot be shared between two SWF files.

To assign the object to a variable, use syntax like the following

```
var so:SharedObject = SharedObject.getLocal("savedData")
```

Because the data may not be immediately available for reading on the device, the application must create and register a listener for the shared object identified by `name`. For details, see the description of the `addListener()` method.

Availability
Flash Lite 2.0

Parameters
name : String - A string that represents the name of the object. The name can include forward slashes (`/`); for example, `work/addresses` is a valid name. Spaces are not allowed in a shared object name, nor are the following characters:

```
~ % & \ ; : " ' , < > ? #
```

Returns
SharedObject - A reference to a shared object that is persistent locally and is available only to the current client. If Flash can't create or find the shared object, `getLocal()` returns `null`.

This method fails and returns `null` if persistent shared object creation and storage by third-party Flash content is prohibited by the device.

Example
The following example saves the last frame that a user entered to a local shared object named `kookie`:

```
// Get the kookie
var my_so:SharedObject = SharedObject.getLocal("kookie");

// Get the user of the kookie and go to the frame number saved for this user.
if (my_so.data.user != undefined) {
    this.user = my_so.data.user;
    this.gotoAndStop(my_so.data.frame);
}
```

The following code block is placed on each SWF file frame:

```
// On each frame, call the rememberme function to save the frame number.
function rememberme() {
    my_so.data.frame=this._currentframe;
    my_so.data.user="John";
}
```

getMaxSize (SharedObject.getMaxSize method)

```
public static getMaxSize() : Number
```

Returns the total number of bytes the SWF file can use to store mobile shared objects on the device.

For example, if this method returns 1K, the movie can save one shared object of 1K, or multiple smaller shared objects, as long as their combined size does not exceed 1K. This method is a static method of the SharedObject class.

Availability
Flash Lite 2.0

Returns

Number - A numeric value that specifies the total number of bytes the movie is allowed to store on the device. This is also the size available to all content that is loaded dynamically through `loadMovie()`.

Example

The following example checks whether more than 1KB of storage is reserved before creating a Flash Lite shared object.

```
if (SharedObject.getMaxSize() > 1024) {
    var my_so:SharedObject = SharedObject.getLocal("sharedObject1");
} else {
    trace("SharedObject's maximum size is less than 1 KB.");
}
```

getSize (SharedObject.getSize method)

```
public getSize() : Number
```

Gets the current size of the shared object, in bytes.

Flash calculates the size of a shared object by stepping through all of its data properties; the more data properties the object has, the longer it takes to estimate its size. Estimating object size can take significant processing time, so you may want to avoid using this method unless you have a specific need for it.

If the shared object listener has not yet been called, `getSize()` returns 0. For details about using the listener, see the `addListener()` method.

Availability

Flash Lite 2.0

Returns

Number - A numeric value specifying the size of the shared object, in bytes.

Example

The following example gets the size of the shared object `my_so`:

```
var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

var soSize:Number = my_so.getSize();
trace(soSize);
```

onStatus (SharedObject.onStatus handler)

```
onStatus = function(infoObject:Object) {}
```

Invoked every time an error, warning, or informational note is posted for a shared object. To respond to this event handler, you must create a function to process the information object that is generated by the shared object.

The information object has a code property that contains a string that describes the result of the `onStatus` handler, and a `level` property that contains a string that is either `"Status"` or `"Error"`.

In addition to this handler, Flash Lite also provides a super function called `System.onStatus`. If `onStatus` is invoked for a particular object and no function is assigned to respond to it, Flash Lite processes a function assigned to `System.onStatus`, if it exists.

The following events notify you when certain SharedObject activities occur:

Code property	Level property	Meaning
`SharedObject.Flush.Failed`	Error	`SharedObject.flush()` method that returned `"pending"` has failed (the user did not allot additional disk space for the shared object when Flash Lite player showed the Local Storage Settings dialog box).
`SharedObject.Flush.Success`	Status	`SharedObject.flush()` method that returned `"pending"` was successfully completed (the user allotted additional disk space for the shared object).

Availability

Flash Lite 2.0

Parameters

infoObject : Object - A parameter defined according to the status message.

Example

The following example displays different messages based on whether the user chooses to allow or deny the SharedObject instance to write to the disk.

```
this.createTextField("message_txt", this.getNextHighestDepth(), 0, 30, 120, 50);
this.message_txt.wordWrap = true;

this.createTextField("status_txt", this.getNextHighestDepth(), 0, 90, 120, 100);
this.status_txt.wordWrap = true;

var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

my_so.onStatus = function(infoObject:Object) {
    for (var i in infoObject) {
    status_txt.text += i+"-"+infoObject[i] +"\n";
    }
};

var flushResult = my_so.flush(1000001);
switch (flushResult) {
    case 'pending' :
    message_txt.text = "flush is pending, waiting on user interaction.";
    break;
    case true :
    message_txt.text = "flush was successful. Requested storage space approved.";
    break;
    case false :
    message_txt.text = "flush failed. User denied request for additional storage.";
    break;
}
```

See also

`onStatus (System.onStatus handler)`

removeListener (SharedObject.removeListener method)

`public static removeListener(objectName:String)`

Removes any listeners that were added using the `addListener()` method.

Availability

Flash Lite 2.0

Parameters

objectName: String - A string that represents the name of the shared object.

Sound

```
Object
    |
    +-Sound
```

```
public class Sound
extends Object
```

The Sound class lets you control sound in a movie. You can add sounds to a movie clip from the library while the movie is playing and control those sounds. If you do not specify a target when you create a new Sound object, you can use the methods to control sound for the whole movie.

You must use the constructor `new Sound` to create a Sound object before calling the methods of the Sound class.

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
	duration : Number [read-only]	The duration of a sound, in milliseconds.
	id3 : Object [read-only]	Provides access to the metadata that is part of an MP3 file.
	position : Number [read-only]	The number of milliseconds a sound has been playing.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Event summary

Event	Description
onID3 = function() {}	Invoked each time new ID3 data is available for an MP3 file that you load using `Sound.attachSound()` or `Sound.loadSound()`.
onLoad = function(success:Boolean) {}	Invoked automatically when a sound loads.
onSoundComplete = function() {}	Invoked automatically when a sound finishes playing.

Constructor summary

Signature	Description
Sound([target:Object])	Creates a new Sound object for a specified movie clip.

Method summary

Modifiers	Signature	Description
	attachSound(id:String) : Void	Attaches the sound specified in the id parameter to the specified Sound object.
	getBytesLoaded() : Number	Returns the number of bytes loaded (streamed) for the specified Sound object.
	getBytesTotal() : Number	Returns the size, in bytes, of the specified Sound object.
	getPan() : Number	Returns the pan level set in the last setPan() call as an integer from -100 (left) to +100 (right).
	getTransform() : Object	Returns the sound transform information for the specified Sound object set with the last Sound.setTransform() call.
	getVolume() : Number	Returns the sound volume level as an integer from 0 to 100, where 0 is off and 100 is full volume.
	loadSound(url:String, isStreaming:Boolean) : Void	Loads an MP3 file into a Sound object.
	setPan(value:Number) : Void	Determines how the sound is played in the left and right channels (speakers).
	setTransform(transformObject:Object) : Void	Sets the sound transform (or balance) information, for a Sound object.
	setVolume(value:Number) : Void	Sets the volume for the Sound object.
	start([secondOffset:Number], [loops:Number]) : Void	Starts playing the last attached sound from the beginning if no parameter is specified, or starting at the point in the sound specified by the secondOffset parameter.
	stop([linkageID:String]) : Void	Stops all sounds currently playing if no parameter is specified, or just the sound specified in the idName parameter.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

attachSound (Sound.attachSound method)

```
public attachSound(id:String) : Void
```

Attaches the sound specified in the id parameter to the specified Sound object. The sound must be in the library of the current SWF file and specified for export in the Linkage Properties dialog box. You must call Sound.start() to start playing the sound.

To make sure that the sound can be controlled from any scene in the SWF file, place the sound on the main Timeline of the SWF file.

Availability
Flash Lite 2.0

Parameters
id : String - The identifier of an exported sound in the library. The identifier is located in the Linkage Properties dialog box.

Example
The following example attaches the sound `logoff_id` to `my_sound`. A sound in the library has the linkage identifier `logoff_id`.

```
var my_sound:Sound = new Sound();
my_sound.attachSound("logoff_id");
my_sound.start();
```

duration (Sound.duration property)

```
public duration : Number [read-only]
```

The duration of a sound, in milliseconds.

Note: Flash Lite 2.0 supports this property for native Flash sound only. The sound formats that are specific to a host device are not supported.

Availability
Flash Lite 2.0

Example
The following example loads a sound and displays the duration of the sound file in the Output panel. Add the following ActionScript to your FLA or AS file:

```
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    var totalSeconds:Number = this.duration/1000;
    trace(this.duration+" ms ("+Math.round(totalSeconds)+" seconds)");
    var minutes:Number = Math.floor(totalSeconds/60);
    var seconds = Math.floor(totalSeconds)%60;
    if (seconds<10) {
    seconds = "0"+seconds;
    }
    trace(minutes+":"+seconds);
};
my_sound.loadSound("song1.mp3", true);
```

The following example loads several songs into a SWF file. A progress bar, created using the Drawing API, displays the loading progress. When the music starts and completes loading, information displays in the Output panel. When the music starts and completes loading, information writes to the log file. Add the following ActionScript to your FLA or AS file:

```
var pb_height:Number = 10;
var pb_width:Number = 100;
var pb:MovieClip = this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
pb.createEmptyMovieClip("bar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("vBar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("stroke_mc", pb.getNextHighestDepth());
pb.createTextField("pos_txt", pb.getNextHighestDepth(), 0, pb_height, pb_width, 22);

pb._x = 100;
pb._y = 100;

with (pb.bar_mc) {
    beginFill(0x00FF00);
    moveTo(0, 0);
    lineTo(pb_width, 0);
    lineTo(pb_width, pb_height);
    lineTo(0, pb_height);
    lineTo(0, 0);
    endFill();
    _xscale = 0;
}
with (pb.vBar_mc) {
    lineStyle(1, 0x000000);
    moveTo(0, 0);
    lineTo(0, pb_height);
}
with (pb.stroke_mc) {
    lineStyle(3, 0x000000);
    moveTo(0, 0);
    lineTo(pb_width, 0);
    lineTo(pb_width, pb_height);
    lineTo(0, pb_height);
    lineTo(0, 0);
}

var my_interval:Number;
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    if (success) {
    trace("sound loaded");
    }
};
my_sound.onSoundComplete = function() {
    clearInterval(my_interval);
    trace("Cleared interval");
}
my_sound.loadSound("song3.mp3", true);
my_interval = setInterval(updateProgressBar, 100, my_sound);

function updateProgressBar(the_sound:Sound):Void {
    var pos:Number = Math.round(the_sound.position/the_sound.duration 100);
    pb.bar_mc._xscale = pos;
    pb.vBar_mc._x = pb.bar_mc._width;
    pb.pos_txt.text = pos+"%";
}
```

See also

position (Sound.position property)

getBytesLoaded (Sound.getBytesLoaded method)

```
public getBytesLoaded() : Number
```

Returns the number of bytes loaded (streamed) for the specified Sound object. You can compare the value of `getBytesLoaded()` with the value of `getBytesTotal()` to determine what percentage of a sound has loaded.

Availability

Flash Lite 2.0

Returns

Number - An integer indicating the number of bytes loaded.

Example

The following example dynamically creates two text fields that display the bytes that are loaded and the total number of bytes for a sound file that loads into the SWF file. A text field also displays a message when the file finishes loading. Add the following ActionScript to your FLA or AS file:

```
this.createTextField("message_txt", this.getNextHighestDepth(), 10,10,300,22)
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 50, 300, 40);
status_txt.autoSize = true;
status_txt.multiline = true;
status_txt.border = false;

var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    if (success) {
    this.start();
    message_txt.text = "Finished loading";
    }
};
my_sound.onSoundComplete = function() {
    message_txt.text = "Clearing interval";
    clearInterval(my_interval);
};
my_sound.loadSound("song2.mp3", true);
var my_interval:Number;
my_interval = setInterval(checkProgress, 100, my_sound);
function checkProgress(the_sound:Sound):Void {
    var pct:Number = Math.round(the_sound.getBytesLoaded()/the_sound.getBytesTotal() 100);
    var pos:Number = Math.round(the_sound.position/the_sound.duration 100);
    status_txt.text = the_sound.getBytesLoaded()+" of "+the_sound.getBytesTotal()+" bytes
("+pct+"%)"+newline;
    status_txt.text += the_sound.position+" of "+the_sound.duration+" milliseconds
("+pos+"%)"+newline;
}
```

See also

getBytesTotal (Sound.getBytesTotal method)

getBytesTotal (Sound.getBytesTotal method)

```
public getBytesTotal() : Number
```

Returns the size, in bytes, of the specified Sound object.

Availability
Flash Lite 2.0

Returns
Number - An integer indicating the total size, in bytes, of the specified Sound object.

Example
For a sample usage of this method, see `Sound.getBytesLoaded()`.

See also
`getBytesLoaded (Sound.getBytesLoaded method)`

getPan (Sound.getPan method)

```
public getPan() : Number
```

Returns the pan level set in the last `setPan()` call as an integer from -100 (left) to +100 (right). (0 sets the left and right channels equally.) The pan setting controls the left-right balance of the current and future sounds in a SWF file.

This method is cumulative with `setVolume()` or `setTransform()`.

Note: Flash Lite 2.0 supports this method for native Flash sound only. The sound formats that are specific to a host device are not supported.

Availability
Flash Lite 2.0

Returns
Number - An integer.

Example
The following example creates a text field to display the value of the pan level for native Flash sound. The linkage identifier for the sound is "combo". Add the following ActionScript to your FLA or AS file:

```
this.createTextField("pan_txt", 1, 0, 100, 100, 100);
mix=new Sound();
mix.attachSound("combo");
mix.start();
mix.setPan(-100);
pan_txt.text = mix.getPan(this);
```

You can use the following example to start the device sound. Because Flash Lite does not support streaming sound, it is a good practice to load the sound before playing it.

```
var my_sound:Sound = new Sound();
    my_sound.onLoad = function(success) {
        if (success) {
            my_sound.start();
        } else {
            output.text = "loading failure";
        }
    };
    my_sound.loadSound("song1.mp3",false);
```

See also

setPan (Sound.setPan method)

getTransform (Sound.getTransform method)

```
public getTransform() : Object
```

Returns the sound transform information for the specified Sound object set with the last `Sound.setTransform()` call.

Note: Flash Lite 2.0 supports this method for native Flash sound only. The sound formats that are specific to a host device are not supported.

Availability

Flash Lite 2.0

Returns

Object - An object with properties that contain the channel percentage values for the specified Sound object.

Example

The following example attaches four movie clips from a symbol in the library (linkage identifier: `knob_id`) that are used as sliders (or knobs) to control the sound file that loads into the SWF file. These sliders control the transform object, or balance, of the sound file. For more information, see the entry for `Sound.setTransform()`. Add the following ActionScript to your FLA or AS file:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
var transform_obj:Object = my_sound.getTransform();

this.createEmptyMovieClip("transform_mc", this.getNextHighestDepth());
transform_mc.createTextField("transform_txt", transform_mc.getNextHighestDepth, 0, 8, 120, 22);
transform_mc.transform_txt.html = true;

var knob_ll:MovieClip = transform_mc.attachMovie("knob_id", "ll_mc",
transform_mc.getNextHighestDepth(), {_x:0, _y:30});
var knob_lr:MovieClip = transform_mc.attachMovie("knob_id", "lr_mc",
transform_mc.getNextHighestDepth(), {_x:30, _y:30});
var knob_rl:MovieClip = transform_mc.attachMovie("knob_id", "rl_mc",
transform_mc.getNextHighestDepth(), {_x:60, _y:30});
var knob_rr:MovieClip = transform_mc.attachMovie("knob_id", "rr_mc",
transform_mc.getNextHighestDepth(), {_x:90, _y:30});

knob_ll.top = knob_ll._y;
knob_ll.bottom = knob_ll._y+100;
knob_ll.left = knob_ll._x;
```

```
knob_ll.right = knob_ll._x;
knob_ll._y = knob_ll._y+(100-transform_obj['ll']);
knob_ll.onPress = pressKnob;
knob_ll.onRelease = releaseKnob;
knob_ll.onReleaseOutside = releaseKnob;

knob_lr.top = knob_lr._y;
knob_lr.bottom = knob_lr._y+100;
knob_lr.left = knob_lr._x;
knob_lr.right = knob_lr._x;
knob_lr._y = knob_lr._y+(100-transform_obj['lr']);
knob_lr.onPress = pressKnob;
knob_lr.onRelease = releaseKnob;
knob_lr.onReleaseOutside = releaseKnob;

knob_rl.top = knob_rl._y;
knob_rl.bottom = knob_rl._y+100;
knob_rl.left = knob_rl._x;
knob_rl.right = knob_rl._x;
knob_rl._y = knob_rl._y+(100-transform_obj['rl']);
knob_rl.onPress = pressKnob;
knob_rl.onRelease = releaseKnob;
knob_rl.onReleaseOutside = releaseKnob;

knob_rr.top = knob_rr._y;
knob_rr.bottom = knob_rr._y+100;
knob_rr.left = knob_rr._x;
knob_rr.right = knob_rr._x;
knob_rr._y = knob_rr._y+(100-transform_obj['rr']);
knob_rr.onPress = pressKnob;
knob_rr.onRelease = releaseKnob;

knob_rr.onReleaseOutside = releaseKnob;

updateTransformTxt();

function pressKnob() {
    this.startDrag(false, this.left, this.top, this.right, this.bottom);
}
function releaseKnob() {
    this.stopDrag();
    updateTransformTxt();
}
function updateTransformTxt() {
    var ll_num:Number = 30+100-knob_ll._y;
    var lr_num:Number = 30+100-knob_lr._y;
    var rl_num:Number = 30+100-knob_rl._y;
    var rr_num:Number = 30+100-knob_rr._y;
    my_sound.setTransform({ll:ll_num, lr:lr_num, rl:rl_num, rr:rr_num});
    transform_mc.transform_txt.htmlText = "<textformat tabStops='[0,30,60,90]'>";
    transform_mc.transform_txt.htmlText += ll_num+"\t"+lr_num+"\t"+rl_num+"\t"+rr_num;
    transform_mc.transform_txt.htmlText += "</textformat>";
}
```

See also

setTransform (Sound.setTransform method)

getVolume (Sound.getVolume method)

```
public getVolume() : Number
```

Returns the sound volume level as an integer from 0 to 100, where 0 is off and 100 is full volume. The default setting is 100.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example creates a slider using the Drawing API and a movie clip that is created at runtime. A dynamically created text field displays the current volume level of the sound playing in the SWF file. Add the following ActionScript to your ActionScript or FLA file:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song3.mp3", true);

this.createEmptyMovieClip("knob_mc", this.getNextHighestDepth());

knob_mc.left = knob_mc._x;
knob_mc.right = knob_mc.left+100;
knob_mc.top = knob_mc._y;
knob_mc.bottom = knob_mc._y;

knob_mc._x = my_sound.getVolume();

with (knob_mc) {
    lineStyle(0, 0x000000);
    beginFill(0xCCCCCC);
    moveTo(0, 0);
    lineTo(4, 0);
    lineTo(4, 18);
    lineTo(0, 18);
    lineTo(0, 0);
    endFill();
```

```
}

knob_mc.createTextField("volume_txt", knob_mc.getNextHighestDepth(), knob_mc._width+4, 0, 30, 22);
knob_mc.volume_txt.text = my_sound.getVolume();

knob_mc.onPress = function() {
    this.startDrag(false, this.left, this.top, this.right, this.bottom);
    this.isDragging = true;
};
knob_mc.onMouseMove = function() {
    if (this.isDragging) {
    this.volume_txt.text = this._x;
    }
}
knob_mc.onRelease = function() {
    this.stopDrag();
    this.isDragging = false;
    my_sound.setVolume(this._x);

};
```

See also

setVolume (Sound.setVolume method)

id3 (Sound.id3 property)

```
public id3 : Object [read-only]
```

Provides access to the metadata that is part of an MP3 file.

MP3 sound files can contain ID3 tags, which provide metadata about the file. If an MP3 sound that you load using `Sound.attachSound()` or `Sound.loadSound()` contains ID3 tags, you can query these properties. Only ID3 tags that use the UTF-8 character set are supported.

Flash Player 6 (6.0.40.0) and later use the `Sound.id3` property to support ID3 1.0 and ID3 1.1 tags. Flash Player 7 adds support for ID3 2.0 tags, specifically 2.3 and 2.4. The following table lists the standard ID3 2.0 tags and the type of content the tags represent; you query them in the format *my_sound*`.id3.COMM`, *my_sound*`.id3.TIME`, and so on. MP3 files can contain tags other than those in this table; `Sound.id3` provides access to those tags as well.

Property	Description
TFLT	File type
TIME	Time
TIT1	Content group description
TIT2	Title/song name/content description
TIT3	Subtitle/description refinement
TKEY	Initial key
TLAN	Languages
TLEN	Length
TMED	Media type

Property	Description
TOAL	Original album/movie/show title
TOFN	Original filename
TOLY	Original lyricists/text writers
TOPE	Original artists/performers
TORY	Original release year
TOWN	File owner/licensee
TPE1	Lead performers/soloists
TPE2	Band/orchestra/accompaniment
TPE3	Conductor/performer refinement
TPE4	Interpreted, remixed, or otherwise modified by
TPOS	Part of a set
TPUB	Publisher
TRCK	Track number/position in set
TRDA	Recording dates
TRSN	Internet radio station name
TRSO	Internet radio station owner
TSIZ	Size
TSRC	ISRC (international standard recording code)
TSSE	Software/hardware and settings used for encoding
TYER	Year
WXXX	URL link frame

Flash Player 6 supported several ID31.0 tags. If these tags are in not in the MP3 file, but corresponding ID3 2.0 tags are, the ID3 2.0 tags are copied into the ID3 1.0 properties, as shown in the following table. This process provides backward compatibility with scripts that you may have written already that read ID3 1.0 properties.

ID3 2.0 tag	Corresponding ID3 1.0 property
COMM	Sound.id3.comment
TALB	Sound.id3.album
TCON	Sound.id3.genre
TIT2	Sound.id3.songname
TPE1	Sound.id3.artist
TRCK	Sound.id3.track
TYER	Sound.id3.year

Availability

Flash Lite 2.0

Example

The following example traces the ID3 properties of song.mp3 to the Output panel:

```
var my_sound:Sound = new Sound();
my_sound.onID3 = function(){
    for( var prop in my_sound.id3 ){
    trace( prop + " : "+ my_sound.id3[prop] );
    }
}
my_sound.loadSound("song.mp3", false);
```

See also

attachSound (Sound.attachSound method), loadSound (Sound.loadSound method)

loadSound (Sound.loadSound method)

```
public loadSound(url:String, isStreaming:Boolean) : Void
```

Loads an MP3 file into a Sound object. You can use the `isStreaming` parameter to indicate whether the sound is an event or a streaming sound.

Event sounds are completely loaded before they play. They are managed by the ActionScript Sound class and respond to all methods and properties of this class.

Streaming sounds play while they are downloading. Playback begins when sufficient data is received to start the decompressor.

All MP3s (event or streaming) loaded with this method are saved in the browser's file cache on the user's system.

Note: For Flash Lite 2.0, you can ignore the `isStreaming` parameter because Flash Lite 2.0 treats every sound as an event sound.

Availability

Flash Lite 2.0

Parameters

url : String - The location on a server of an MP3 sound file.

isStreaming : Boolean - A Boolean value that indicates whether the sound is a streaming sound (`true`) or an event sound (`false`).

Example

The following example loads an event sound, which cannot play until it is fully loaded:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", false);
```

The following example loads a streaming sound:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
```

See also

onLoad (Sound.onLoad handler)

onID3 (Sound.onID3 handler)

```
onID3 = function() {}
```

Invoked each time new ID3 data is available for an MP3 file that you load using `Sound.attachSound()` or `Sound.loadSound()`. This handler provides access to ID3 data without polling. If both ID3 1.0 and ID3 2.0 tags are present in a file, this handler is called twice.

Availability

Flash Lite 2.0

Example

The following example displays the ID3 properties of `song1.mp3` to an instance of the DataGrid component. Add a DataGrid with the instance name `id3_dg` to your document, and add the following ActionScript to your FLA or AS file:

```
import mx.controls.gridclasses.DataGridColumn;
var id3_dg:mx.controls.DataGrid;
id3_dg.move(0, 0);
id3_dg.setSize(Stage.width, Stage.height);
var property_dgc:DataGridColumn = id3_dg.addColumn(new DataGridColumn("property"));
property_dgc.width = 100;
property_dgc.headerText = "ID3 Property";
var value_dgc:DataGridColumn = id3_dg.addColumn(new DataGridColumn("value"));
value_dgc.width = id3_dg._width-property_dgc.width;
value_dgc.headerText = "ID3 Value";

var my_sound:Sound = new Sound();
my_sound.onID3 = function() {
trace("onID3 called at "+getTimer()+" ms.");
for (var prop in this.id3) {
id3_dg.addItem({property:prop, value:this.id3[prop]});
}
};
my_sound.loadSound("song1.mp3", true);
```

See also

attachSound (Sound.attachSound method), id3 (Sound.id3 property), loadSound (Sound.loadSound method)

onLoad (Sound.onLoad handler)

```
onLoad = function(success:Boolean) {}
```

Invoked automatically when a sound loads. You must create a function that executes when the this handler is invoked. You can use either an anonymous function or a named function (for an example of each, see `Sound.onSoundComplete`). You should define this handler before you call `mySound.loadSound()`.

Availability

Flash Lite 2.0

Parameters

success : Boolean - A Boolean value of `true` if `my_sound` is loaded successfully, `false` otherwise.

Example

The following example creates a new Sound object, and loads a sound. Loading the sound is handled by the `onLoad` handler, which allows you to start the song after it is successfully loaded. Create a new FLA file, and add the following ActionScript to your FLA or AS file. For this example to work, you must have an MP3 called `song1.mp3` in the same directory as your FLA or AS file.

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);

// create a new Sound object
var my_sound:Sound = new Sound();
// If the sound loads, play it; if not, trace failure loading.
my_sound.onLoad = function(success:Boolean) {
if (success) {
my_sound.start();
status_txt.text = "Sound loaded";
} else {
status_txt.text = "Sound failed";
}
};
// Load the sound.
my_sound.loadSound("song1.mp3", true);
```

See also

`loadSound (Sound.loadSound method)`

onSoundComplete (Sound.onSoundComplete handler)

```
onSoundComplete = function() {}
```

Invoked automatically when a sound finishes playing. You can use this handler to trigger events in a SWF file when a sound finishes playing.

You must create a function that executes when this handler is invoked. You can use either an anonymous function or a named function.

Availability

Flash Lite 2.0

Example

Usage 1: The following example uses an anonymous function:

```
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = function() {
trace("mySoundID completed");
};
my_sound.start();
```

Usage 2: The following example uses a named function:

```
function callback1() {
trace("mySoundID completed");
}
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = callback1;
my_sound.start();
```

See also

onLoad (Sound.onLoad handler)

position (Sound.position property)

```
public position : Number [read-only]
```

The number of milliseconds a sound has been playing. If the sound is looped, the position is reset to 0 at the beginning of each loop.

Note: Flash Lite 2.0 supports this property for native Flash sound only. The sound formats that are specific to a host device are not supported.

Availability

Flash Lite 2.0

Example

For a sample usage of this property, see `Sound.duration`.

See also

duration (Sound.duration property)

setPan (Sound.setPan method)

```
public setPan(value:Number) : Void
```

Determines how the sound is played in the left and right channels (speakers). For mono sounds, *pan* determines which speaker (left or right) the sound plays through.

Note: Flash Lite 2.0 supports this method for native Flash sound only. The sound formats that are specific to a host device are not supported.

Availability

Flash Lite 2.0

Parameters

value : Number - An integer specifying the left-right balance for a sound. The range of valid values is -100 to 100, where -100 uses only the left channel, 100 uses only the right channel, and 0 balances the sound evenly between the two channels.

Example

For a sample usage of this method, see `Sound.getPan()`.

See also

attachSound (Sound.attachSound method), getPan (Sound.getPan method), setTransform (Sound.setTransform method) setVolume (Sound.setVolume method), start (Sound.start method)

setTransform (Sound.setTransform method)

```
public setTransform(transformObject:Object) : Void
```

Sets the sound transform (or balance) information, for a Sound object.

The `soundTransformObject` parameter is an object that you create using the constructor method of the generic Object class with parameters specifying how the sound is distributed to the left and right channels (speakers).

Sounds use a considerable amount of disk space and memory. Because stereo sounds use twice as much data as mono sounds, it is generally best to use 22-KHz 6-bit mono sounds. You can use `setTransform()` to play mono sounds as stereo, play stereo sounds as mono, and to add interesting effects to sounds.

Note: Flash Lite 2.0 supports this method for native Flash sound only. The sound formats that are specific to a host device are not supported.

The properties for the `soundTransformObject` are as follows:

`ll` - A percentage value that specifies how much of the left input to play in the left speaker (0-100).

`lr` - A percentage value that specifies how much of the right input to play in the left speaker (0-100).

`rr` - A percentage value that specifies how much of the right input to play in the right speaker (0-100).

`rl` - A percentage value that specifies how much of the left input to play in the right speaker (0-100).

The net result of the parameters is represented by the following formula:

```
leftOutput = left_input ~ ll + right_input ~ lr
rightOutput = right_input ~ rr + left_input ~ rl
```

The values for `left_input` or `right_input` are determined by the type (stereo or mono) of sound in your SWF file.

Stereo sounds divide the sound input evenly between the left and right speakers and have the following transform settings by default:

```
ll = 100
lr = 0
rr = 100
rl = 0
```

Mono sounds play all sound input in the left speaker and have the following transform settings by default:

```
ll = 100
lr = 100
rr = 0
rl = 0
```

Availability

Flash Lite 2.0

Parameters

transformObject : Object - An object created with the constructor for the generic Object class.

Example

The following example illustrates a setting that can be achieved by using `setTransform()`, but cannot be achieved by using `setVolume()` or `setPan()`, even if they are combined.

The following code creates a new soundTransformObject object and sets its properties so that sound from both channels plays in the left channel only .

```
var mySoundTransformObject:Object = new Object();
mySoundTransformObject.ll = 100;
mySoundTransformObject.lr = 100;
mySoundTransformObject.rr = 0;
mySoundTransformObject.rl = 0;
```

To apply the soundTransformObject object to a Sound object, you then need to pass the object to the Sound object using `setTransform()` as follows:

```
my_sound.setTransform(mySoundTransformObject);
```

The following example plays a stereo sound as mono; the soundTransformObjectMono object has the following parameters:

```
var mySoundTransformObjectMono:Object = new Object();
mySoundTransformObjectMono.ll = 50;
mySoundTransformObjectMono.lr = 50;
mySoundTransformObjectMono.rr = 50;
mySoundTransformObjectMono.rl = 50;
my_sound.setTransform(mySoundTransformObjectMono);
```

This example plays the left channel at half capacity and adds the rest of the left channel to the right channel; the soundTransformObjectHalf object has the following parameters:

```
var mySoundTransformObjectHalf:Object = new Object();
mySoundTransformObjectHalf.ll = 50;
mySoundTransformObjectHalf.lr = 0;
mySoundTransformObjectHalf.rr = 100;
mySoundTransformObjectHalf.rl = 50;
my_sound.setTransform(mySoundTransformObjectHalf);

var mySoundTransformObjectHalf:Object = {ll:50, lr:0, rr:100, rl:50};
```

See also

`Object`, `getTransform (Sound.getTransform method)`

setVolume (Sound.setVolume method)

```
public setVolume(value:Number) : Void
```

Sets the volume for the Sound object.

Availability

Flash Lite 2.0

Parameters

value : Number - A number from 0 to 100 representing a volume level. 100 is full volume and 0 is no volume. The default setting is 100.

Example

For a sample usage of this method, see `Sound.getVolume()`.

See also

`setPan (Sound.setPan method)`, `setTransform (Sound.setTransform method)`

Sound constructor

```
public Sound([target:Object])
```

Creates a new Sound object for a specified movie clip. If you do not specify a target instance, the Sound object controls all of the sounds in the movie.

Availability

Flash Lite 2.0

Parameters

target : Object [optional] - The movie clip instance on which the Sound object operates.

Example

The following example creates a new Sound object called `global_sound`. The second line calls `setVolume()` and adjusts the volume on all sounds in the movie to 50%.

```
var global_sound:Sound = new Sound();
global_sound.setVolume(50);
```

The following example creates a new Sound object, passes it the target movie clip *my_mc*, and calls the `start` method, which starts any sound in *my_mc*.

```
var movie_sound:Sound = new Sound(my_mc);
movie_sound.start();
```

start (Sound.start method)

```
public start([secondOffset:Number], [loops:Number]) : Void
```

Starts playing the last attached sound from the beginning if no parameter is specified, or starting at the point in the sound specified by the `secondOffset` parameter.

Availability

Flash Lite 2.0

Parameters

secondOffset : Number [optional] - A parameter that lets you start playing the sound at a specific point. For example, if you have a 30-second sound and want the sound to start playing in the middle, specify 15 for the `secondOffset` parameter. The sound is not delayed 15 seconds, but rather starts playing at the 15-second mark.

loops : Number [optional] - A parameter that lets you specify the number of times the sound should play consecutively. This parameter is not available if the sound is a streaming sound.

Example

The following example creates a new Sound object, and loads a sound. The `onLoad` handler loads the sound, which allows you to start the song after it is successfully loaded. Then the sound uses the `start()` method to start playing. Create a new FLA file, and add the following ActionScript to your FLA or ActionScript file. For this example to work, you must have an MP3 called song1.mp3 in the same directory as your FLA or AS file.

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);

// create a new Sound object
var my_sound:Sound = new Sound();
// If the sound loads, play it; if not, trace failure loading.
my_sound.onLoad = function(success:Boolean) {
    if (success) {
    my_sound.start();
    status_txt.text = "Sound loaded";
    } else {
    status_txt.text = "Sound failed";
    }
};
// Load the sound.
my_sound.loadSound("song1.mp3", true);
```

See also

`stop (Sound.stop method)`

stop (Sound.stop method)

```
public stop([linkageID:String]) : Void
```

Stops all sounds currently playing if no parameter is specified, or just the sound specified in the `idName` parameter.

Availability

Flash Lite 2.0

Parameters

linkageID : String [optional] - A parameter specifying a specific sound to stop playing. The `idName` parameter must be enclosed in quotation marks (" ").

Example

The following example uses two buttons, `stop_btn` and `play_btn`, to control the playback of a sound that loads into a SWF file. Add two buttons to your document and add the following ActionScript to your FLA or AS file:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);

stop_btn.onRelease = function() {
    trace("sound stopped");
    my_sound.stop();
};
play_btn.onRelease = function() {
    trace("sound started");
    my_sound.start();
};
```

See also

start (Sound.start method)

Stage

```
Object
    |
    +-Stage
```

```
public class Stage
extends Object
```

The Stage class is a top-level class whose methods, properties, and handlers you can access without using a constructor. Use the methods and properties of this class to access and manipulate information about the boundaries of a SWF file.

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
static	align : String	Indicates the current alignment of the SWF file in the player or browser.
static	height : Number	Property (read-only); indicates the current height, in pixels, of the Stage.
static	scaleMode : String	Indicates the current scaling of the SWF file within Flash Lite player.
static	width : Number	Property (read-only); indicates the current width, in pixels, of the Stage.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Event summary

Event	Description
onResize = function() {}	Invoked when Stage.scaleMode is set to noScale and the SWF file is resized.

Method summary

Modifiers	Signature	Description
static	addListener(listener: Object) : Void	Detects when a SWF file is resized (but only if Stage.scaleMode = "noScale").
static	removeListener(listener:Object) : Boolean	Removes a listener object created with addListener().

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

addListener (Stage.addListener method)

```
public static addListener(listener:Object) : Void
```

Detects when a SWF file is resized (but only if `Stage.scaleMode = "noScale"`). The `addListener()` method doesn't work with the default movie clip scaling setting (`showAll`) or other scaling settings (`exactFit` and `noBorder`).

To use `addListener()`, you must first create a *listener object*. Stage listener objects receive notification from `Stage.onResize`.

Availability

Flash Lite 2.0

Parameters

listener : Object - An object that listens for a callback notification from the `Stage.onResize` event.

Example

This example creates a new listener object called `stageListener`. It then uses `stageListener` to call `onResize` and define a function that will be called when `onResize` is triggered. Finally, the code adds the stageListener object to the callback list of the Stage object. Listener objects allow multiple objects to listen for resize notifications.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

See also

onResize (Stage.onResize event listener), removeListener (Stage.removeListener method)

align (Stage.align property)

```
public static align : String
```

Indicates the current alignment of the SWF file in the player or browser.

The following table lists the values for the `align` property. Any value not listed here centers the SWF file in Flash player or browser area, which is the default setting.

Value	Vertical	Horizontal
"T"	top	center
"B"	bottom	center
"L"	center	left
"R"	center	right
"TL"	top	left
"TR"	top	right
"BL"	bottom	left
"BR"	bottom	right

Availability

Flash Lite 2.0

Example

The following example demonstrates different alignments of the SWF file. Add a ComboBox instance to your document with the instance name `stageAlign_cb`. Add the following ActionScript to your FLA or AS file:

```
var stageAlign_cb:mx.controls.ComboBox;
stageAlign_cb.dataProvider = ['T', 'B', 'L', 'R', 'TL', 'TR', 'BL', 'BR'];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
    var align:String = evt.target.selectedItem;
    Stage.align = align;
};
stageAlign_cb.addEventListener("change", cbListener);
Stage.scaleMode = "noScale";
```

Select different alignment settings from the ComboBox.

height (Stage.height property)

```
public static height : Number
```

Property (read-only); indicates the current height, in pixels, of the Stage. When the value of `Stage.scaleMode` is noScale, the height property represents the height of Flash Lite player. When the value of `Stage.scaleMode` is not `noScale`, height represents the height of the SWF file.

Availability

Flash Lite 2.0

Example

This example creates a new listener object called `stageListener`. It then uses `myListener` to call `onResize` and define a function that will be called when `onResize` is triggered. Finally, the code adds the myListener object to the callback list of the Stage object. Listener objects allow multiple objects to listen for resize notifications.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

See also

align (Stage.align property), scaleMode (Stage.scaleMode property), width (Stage.width property)

onResize (Stage.onResize event listener)

```
onResize = function() {}
```

Invoked when `Stage.scaleMode` is set to noScale and the SWF file is resized. You can use this event handler to write a function that lays out the objects on the Stage when a SWF file is resized.

Availability

Flash Lite 2.0

Example

The following example displays a message in the Output panel when the Stage is resized:

```
Stage.scaleMode = "noScale"
var myListener:Object = new Object();
myListener.onResize = function () {
    trace("Stage size is now " + Stage.width + " by " + Stage.height);
}
Stage.addListener(myListener);
// later, call Stage.removeListener(myListener)
```

See also

scaleMode (Stage.scaleMode property), addListener (Stage.addListener method), removeListener (Stage.removeListener method)

removeListener (Stage.removeListener method)

```
public static removeListener(listener:Object) : Boolean
```

Removes a listener object created with `addListener()`.

Availability

Flash Lite 2.0

Parameters

listener : Object - An object added to an object's callback list with `addListener()`.

Returns

Boolean - A Boolean value.

Example

The following example displays the Stage dimensions in a dynamically created text field. When you resize the Stage, the values in the text field update. Create a button with an instance name `remove_btn`. Add the following ActionScript to Frame 1 of the Timeline.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
stageSize_txt.autoSize = true;
stageSize_txt.border = true;
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.addListener(stageListener);

remove_btn.onRelease = function() {
    stageSize_txt.text = "Removing Stage listener...";
    Stage.removeListener(stageListener);
}
```

Select Control > Test Movie to test this example. The values you see in the text field are updated when you resize the testing environment. When you click `remove_btn`, the listener is removed and the values are no longer updated in the text field.

See also

`addListener (Stage.addListener method)`

scaleMode (Stage.scaleMode property)

```
public static scaleMode : String
```

Indicates the current scaling of the SWF file within Flash Lite player. The `scaleMode` property forces the SWF file into a specific scaling mode. By default, the SWF file uses the HTML parameters set in the Publish Settings dialog box.

The `scaleMode` property can use the values `"exactFit"`, `"showAll"`, `"noBorder"`, and `"noScale"`. Any other value sets the `scaleMode` property to the default `"showAll"`.

- `showAll` (Default) makes the entire Flash content visible in the specified area without distortion while maintaining the original aspect ratio. Borders can appear on two sides of the application.
- `noBorder` scales the Flash content to fill the specified area, without distortion but possibly with some cropping, while maintaining the original aspect ratio of the application.
- `exactFit` makes the entire Flash content visible in the specified area without trying to preserve the original aspect ratio. Distortion can occur.
- `noScale` makes the size of the Flash content fixed, so that it remains unchanged even as the size of the player window changes. Cropping may occur if the player window is smaller than the Flash content.

Note: the default setting is showAll, except when in test movie mode, where the default setting is `noScale`

Availability

Flash Lite 2.0

Example

The following example demonstrates various scale settings for the SWF file. Add a ComboBox instance to your document with the instance name `scaleMode_cb`. Add the following ActionScript to your FLA or AS file:

```
var scaleMode_cb:mx.controls.ComboBox;
scaleMode_cb.dataProvider = ["showAll", "exactFit", "noBorder", "noScale"];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
    var scaleMode_str:String = evt.target.selectedItem;
    Stage.scaleMode = scaleMode_str;
};
scaleMode_cb.addEventListener("change", cbListener);
```

To view another example, see the stagesize.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples_and_tutorials. Download and decompress the Samples_and_Tutorials .zip file for your Flash Lite version and navigate to the ActionScript folder to access the sample file.

width (Stage.width property)

```
public static width : Number
```

Property (read-only); indicates the current width, in pixels, of the Stage. When the value of `Stage.scaleMode` is `"noScale"`, the `width` property represents the width of Flash Lite player. This means that `Stage.width` will vary as you resize the player window. When the value of `Stage.scaleMode` is not `"noScale"`, `width` represents the width of the SWF file as set at author-time in the Document Properties dialog box. This means that the value of `width` will stay constant as you resize the player window.

Availability

Flash Lite 2.0

Example

This example creates a new listener object called `stageListener`. It then uses `stageListener` to call `onResize` and define a function that will be called when `onResize` is triggered. Finally, the code adds the stageListener object to the callback list of the Stage object. Listener objects allow multiple objects to listen for resize notifications.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

See also

align (Stage.align property), height (Stage.height property), scaleMode (Stage.scaleMode property)

String

```
Object
    |
    +-String

public class String
extends Object
```

The String class is a wrapper for the string primitive data type, and provides methods and properties that let you manipulate primitive string value types. You can convert the value of any object into a string using the `String()` function.

All the methods of the String class, except for `concat()`, `fromCharCode()`, `slice()`, and `substr()`, are generic, which means the methods call `toString()` before performing their operations, and you can use these methods with other non-String objects.

Because all string indexes are zero-based, the index of the last character for any string `x` is `x.length - 1`.

You can call any of the methods of the String class using the constructor method `new String` or using a string literal value. If you specify a string literal, the ActionScript interpreter automatically converts it to a temporary String object, calls the method, and then discards the temporary String object. You can also use the `String.length` property with a string literal.

Do not confuse a string literal with a String object. In the following example, the first line of code creates the string literal `first_string`, and the second line of code creates the String object `second_string`:

```
var first_string:String = "foo"
var second_string:String = new String("foo")
```

Use string literals unless you specifically need to use a String object.

Availability

Flash Lite 2.0

Property summary

Modifiers	Property	Description
	length : Number	An integer specifying the number of characters in the specified String object.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Constructor summary

Signature	Description
String(value:String)	Creates a new String object.

Method summary

Modifiers	Signature	Description
	charAt(index:Number) : String	Returns the character in the position specified by the parameter `index`.
	charCodeAt(index:Number) : Number	Returns a 16-bit integer from 0 to 65535 that represents the character specified by `index`.
	concat(value:Object) : String	Combines the value of the String object with the parameters and returns the newly formed string; the original value, `my_str`, is unchanged.

Modifiers	Signature	Description
static	fromCharCode() : String	Returns a string comprising the characters represented by the Unicode values in the parameters.
	indexOf(value:String, [startIndex:Number]) : Number	Searches the string and returns the position of the first occurrence of value found at or after startIndex within the calling string.
	lastIndexOf(value:String, [startIndex:Number]) : Number	Searches the string from right to left and returns the index of the last occurrence of value found before startIndex within the calling string.
	slice(start:Number, end:Number) : String	Returns a string that includes the start character and all characters up to, but not including, the end character.
	split(delimiter:String, [limit:Number]) : Array	Splits a String object into substrings by breaking it wherever the specified delimiter parameter occurs and returns the substrings in an array.
	substr(start:Number, length:Number) : String	Returns the characters in a string from the index specified in the start parameter through the number of characters specified in the length parameter.
	substring(start:Number, end:Number) : String	Returns a string comprising the characters between the points specified by the start and end parameters.
	toLowerCase() : String	Returns a copy of the String object, with all uppercase characters converted to lowercase.
	toString() : String	Returns an object's properties as strings regardless of whether the properties are strings.
	toUpperCase() : String	Returns a copy of the String object, with all lowercase characters converted to uppercase.
	valueOf() : String	Returns the string.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

charAt (String.charAt method)

```
public charAt(index:Number) : String
```

Returns the character in the position specified by the parameter index. If index is not a number from 0 to string.length - 1, an empty string is returned.

This method is similar to String.charCodeAt() except that the returned value is a character, not a 16-bit integer character code.

Availability

Flash Lite 2.0

Parameters

index: Number - An integer specifying the position of a character in the string. The first character is indicated by `0`, and the last character is indicated by `my_str.length-1`.

Returns

String - The character at the specified index. Or an empty `String` if the specified index is outside the range of this `String`'s indices.

Example

In the following example, this method is called on the first letter of the string "`Chris`":

```
var my_str:String = "Chris";
var firstChar_str:String = my_str.charAt(0);
trace(firstChar_str); // output: C
```

See also

`charCodeAt (String.charCodeAt method)`

charCodeAt (String.charCodeAt method)

```
public charCodeAt(index:Number) : Number
```

Returns a 16-bit integer from 0 to 65535 that represents the character specified by `index`. If `index` is not a number from 0 to `string.length - 1`, `NaN` is returned.

This method is similar to `String.charAt()` except that the returned value is a 16-bit integer character code, not a character.

Availability

Flash Lite 2.0

Parameters

index: Number - An integer that specifies the position of a character in the string. The first character is indicated by `0`, and the last character is indicated by `my_str.length - 1`.

Returns

Number - An integer that represents the character specified by `index`.

Example

In the following example, this method is called on the first letter of the string "Chris":

```
var my_str:String = "Chris";
var firstChar_num:Number = my_str.charCodeAt(0);
trace(firstChar_num); // output: 67
```

See also

`charAt (String.charAt method)`

concat (String.concat method)

```
public concat(value:Object) : String
```

Combines the value of the String object with the parameters and returns the newly formed string; the original value, `my_str`, is unchanged.

Availability
Flash Lite 2.0

Parameters
value : Object - value1[,...valueN]—Zero or more values to be concatenated.

Returns
String - A string.

Example
The following example creates two strings and combines them using `String.concat()`:

```
var stringA:String = "Hello";
var stringB:String = "World";
var combinedAB:String = stringA.concat(" ", stringB);
trace(combinedAB); // output: Hello World
```

fromCharCode (String.fromCharCode method)

```
public static fromCharCode() : String
```

Returns a string comprising the characters represented by the Unicode values in the parameters.

Availability
Flash Lite 2.0

Returns
String - A string value of the specified Unicode character codes.

Example
The following example uses `fromCharCode()` to insert an `@` character in the e-mail address:

```
var address_str:String = "dog"+String.fromCharCode(64)+"house.net";
trace(address_str); // output: dog@house.net
```

indexOf (String.indexOf method)

```
public indexOf(value:String, [startIndex:Number]) : Number
```

Searches the string and returns the position of the first occurrence of `value` found at or after `startIndex` within the calling string. This index is zero-based, meaning that the first character in a string is considered to be at index 0—not index 1. If `value` is not found, the method returns -1.

Availability
Flash Lite 2.0

Parameters
value : String - A string; the substring to search for.

startIndex: Number [optional] - An integer specifying the starting index of the search.

Returns

Number - The position of the first occurrence of the specified substring or -1.

Example

The following examples use `indexOf()` to return the index of characters and substrings:

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.indexOf("L");
trace(index); // output: 0

index = searchString.indexOf("l");
trace(index); // output: 14

index = searchString.indexOf("i");
trace(index); // output: 6

index = searchString.indexOf("ipsum");
trace(index); // output: 6

index = searchString.indexOf("i", 7);
trace(index); // output: 19

index = searchString.indexOf("z");
trace(index); // output: -1
```

See also

`lastIndexOf (String.lastIndexOf method)`

lastIndexOf (String.lastIndexOf method)

```
public lastIndexOf(value:String, [startIndex:Number]) : Number
```

Searches the string from right to left and returns the index of the last occurrence of `value` found before `startIndex` within the calling string. This index is zero-based, meaning that the first character in a string is considered to be at index 0—not index 1. If `value` is not found, the method returns -1.

Availability

Flash Lite 2.0

Parameters

value: String - The string for which to search.

startIndex: Number [optional] - An integer specifying the starting point from which to search for `value`.

Returns

Number - The position of the last occurrence of the specified substring or -1.

Example

The following example shows how to use `lastIndexOf()` to return the index of a certain character:

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.lastIndexOf("L");
trace(index); // output: 0

index = searchString.lastIndexOf("l");
trace(index); // output: 14

index = searchString.lastIndexOf("i");
trace(index); // output: 19

index = searchString.lastIndexOf("ipsum");
trace(index); // output: 6

index = searchString.lastIndexOf("i", 18);
trace(index); // output: 6

index = searchString.lastIndexOf("z");
trace(index); // output: -1
```

See also

indexOf (String.indexOf method)

length (String.length property)

```
public length : Number
```

An integer specifying the number of characters in the specified String object.

Because all string indexes are zero-based, the index of the last character for any string `x` is `x.length - 1`.

Availability

Flash Lite 2.0

Example

The following example creates a new String object and uses `String.length` to count the number of characters:

```
var my_str:String = "Hello world!";
trace(my_str.length); // output: 12
```

The following example loops from 0 to `my_str.length`. The code checks the characters within a string, and if the string contains the `@` character, `true` displays in the Output panel. If it does not contain the `@` character, then `false` displays in the Output panel.

```
function checkAtSymbol(my_str:String):Boolean {
    for (var i = 0; i<my_str.length; i++) {
    if (my_str.charAt(i) == "@") {
        return true;
    }
    }
    return false;
}

trace(checkAtSymbol("dog@house.net")); // output: true
trace(checkAtSymbol("Chris")); // output: false
```

An example is also in the Strings.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

slice (String.slice method)

```
public slice(start:Number, end:Number) : String
```

Returns a string that includes the `start` character and all characters up to, but not including, the `end` character. The original String object is not modified. If the `end` parameter is not specified, the end of the substring is the end of the string. If the character indexed by `start` is the same as or to the right of the character indexed by `end`, the method returns an empty string.

Availability

Flash Lite 2.0

Parameters

start : Number - The zero-based index of the starting point for the slice. If `start` is a negative number, the starting point is determined from the end of the string, where -1 is the last character.

end : Number - An integer that is one greater than the index of the ending point for the slice. The character indexed by the `end` parameter is not included in the extracted string. If this parameter is omitted, `String.length` is used. If `end` is a negative number, the ending point is determined by counting back from the end of the string, where -1 is the last character.

Returns

String - A substring of the specified string.

Example

The following example creates a variable, `my_str`, assigns it a String value, and then calls the `slice()` method using a variety of values for both the `start` and `end` parameters. Each call to `slice()` is wrapped in a `trace()` statement that displays the output in the Output panel.

```
// Index values for the string literal
// positive index: 0 1 2 3 4
// string: L o r e m
// negative index: -5 -4 -3 -2 -1

var my_str:String = "Lorem";

// slice the first character
trace("slice(0,1): "+my_str.slice(0, 1)); // output: slice(0,1): L
trace("slice(-5,1): "+my_str.slice(-5, 1)); // output: slice(-5,1): L

// slice the middle three characters
trace("slice(1,4): "+my_str.slice(1, 4)); // slice(1,4): ore
trace("slice(1,-1): "+my_str.slice(1, -1)); // slice(1,-1): ore

// slices that return empty strings because start is not to the left of end
trace("slice(1,1): "+my_str.slice(1, 1)); // slice(1,1):
trace("slice(3,2): "+my_str.slice(3, 2)); // slice(3,2):
trace("slice(-2,2): "+my_str.slice(-2, 2)); // slice(-2,2):

// slices that omit the end parameter use String.length, which equals 5
trace("slice(0): "+my_str.slice(0)); // slice(0): Lorem
trace("slice(3): "+my_str.slice(3)); // slice(3): em
```

An example is also in the Strings.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

substr (String.substr method), substring (String.substring method)

split (String.split method)

```
public split(delimiter:String, [limit:Number]) : Array
```

Splits a String object into substrings by breaking it wherever the specified `delimiter` parameter occurs and returns the substrings in an array. If you use an empty string ("") as a delimiter, each character in the string is placed as an element in the array.

If the `delimiter` parameter is undefined, the entire string is placed into the first element of the returned array.

Availability

Flash Lite 2.0

Parameters

delimiter : String - A string; the character or string at which `my_str` splits.

limit : Number [optional] - The number of items to place into the array.

Returns

Array - An array containing the substrings of `my_str`.

Example

The following example returns an array with five elements:

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(",");
for (var i = 0; i<my_array.length; i++) {
    trace(my_array[i]);
}
// output:
    P
    A
    T
    S
    Y
```

The following example returns an array with two elements, `"P"` and `"A"`:

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(",", 2);
trace(my_array); // output: P,A
```

The following example shows that if you use an empty string ("") for the `delimiter` parameter, each character in the string is placed as an element in the array:

```
var my_str:String = new String("Joe");
var my_array:Array = my_str.split("");
for (var i = 0; i<my_array.length; i++) {
    trace(my_array[i]);
}
// output:
    J
    o
    e
```

An example is also in the Strings.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

join (Array.join method)

String constructor

```
public String(value:String)
```

Creates a new String object.

Note: Because string literals use less overhead than String objects and are generally easier to use, you should use string literals instead of the constructor for the String class unless you have a good reason to use a String object rather than a string literal.

Availability

Flash Lite 2.0

Parameters

value : String - The initial value of the new String object.

substr (String.substr method)

```
public substr(start:Number, length:Number) : String
```

Returns the characters in a string from the index specified in the `start` parameter through the number of characters specified in the `length` parameter. The `substr` method does not change the string specified by `my_str`; it returns a new string.

Availability

Flash Lite 2.0

Parameters

start : Number - An integer that indicates the position of the first character in `my_str` to be used to create the substring. If `start` is a negative number, the starting position is determined from the end of the string, where the -1 is the last character.

length : Number - The number of characters in the substring being created. If `length` is not specified, the substring includes all the characters from the start to the end of the string.

Returns

String - A substring of the specified string.

Example

The following example creates a new string, `my_str` and uses `substr()` to return the second word in the string; first, using a positive `start` parameter, and then using a negative `start` parameter:

```
var my_str:String = new String("Hello world");
var mySubstring:String = new String();
mySubstring = my_str.substr(6,5);
trace(mySubstring); // output: world

mySubstring = my_str.substr(-5,5);
trace(mySubstring); // output: world
```

An example is also in the Strings.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

substring (String.substring method)

```
public substring(start:Number, end:Number) : String
```

Returns a string comprising the characters between the points specified by the `start` and `end` parameters. If the `end` parameter is not specified, the end of the substring is the end of the string. If the value of `start` equals the value of `end`, the method returns an empty string. If the value of `start` is greater than the value of `end`, the parameters are automatically swapped before the function executes and the original value is unchanged.

Availability

Flash Lite 2.0

Parameters

start : Number - An integer that indicates the position of the first character of `my_str` used to create the substring. Valid values for `start` are 0 through `String.length` - 1. If `start` is a negative value, 0 is used.

end : Number - An integer that is 1+ the index of the last character in `my_str` to be extracted. Valid values for `end` are 1 through `String.length`. The character indexed by the `end` parameter is not included in the extracted string. If this parameter is omitted, `String.length` is used. If this parameter is a negative value, 0 is used.

Returns

String - A substring of the specified string.

Example

The following example shows how to use `substring()`:

```
var my_str:String = "Hello world";
var mySubstring:String = my_str.substring(6,11);
trace(mySubstring); // output: world
```

The following example shows what happens if a negative `start` parameter is used:

```
var my_str:String = "Hello world";
var mySubstring:String = my_str.substring(-5,5);
trace(mySubstring); // output: Hello
```

An example is also in the Strings.fla file in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

toLowerCase (String.toLowerCase method)

```
public toLowerCase() : String
```

Returns a copy of the String object, with all uppercase characters converted to lowercase. The original value is unchanged.

Availability

Flash Lite 2.0

Returns

String - A string.

Example

The following example creates a string with all uppercase characters and then creates a copy of that string using `toLowerCase()` to convert all uppercase characters to lowercase characters:

```
var upperCase:String = "LOREM IPSUM DOLOR";
var lowerCase:String = upperCase.toLowerCase();
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
```

An example is also in the Strings.fla file in the ActionScript samples folder at The Adobe Flash samples page. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

`toUpperCase (String.toUpperCase method)`

toString (String.toString method)

```
public toString() : String
```

Returns an object's properties as strings regardless of whether the properties are strings.

Availability
Flash Lite 2.0

Returns
String - The string.

Example
The following example outputs an uppercase string that lists all of an object's properties (regardless of whether the properties are strings):

```
var employee:Object = new Object();
employee.name = "bob";
employee.salary = 60000;
employee.id = 284759021;

var employeeData:String = new String();
for (prop in employee)
{
    employeeData += employee[prop].toString().toUpperCase() + " ";
}
trace(employeeData);
```

If the `toString()` method were not included in this code (and the line within the `for` loop used `employee[prop].toUpperCase()`), the output would be "`undefined undefined BOB`". By including the `toString()` method, the desired output is produced: "`284759021 60000 BOB`".

toUpperCase (String.toUpperCase method)

```
public toUpperCase() : String
```

Returns a copy of the String object, with all lowercase characters converted to uppercase. The original value is unchanged.

Availability
Flash Lite 2.0

Returns
String - A string.

Example
The following example creates a string with all lowercase characters and then creates a copy of that string using `toUpperCase()`:

```
var lowerCase:String = "lorem ipsum dolor";
var upperCase:String = lowerCase.toUpperCase();
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
```

An example is also in the Strings.fla file in the ActionScript samples folder at the Adobe Flash samples page. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

toLowerCase (String.toLowerCase method)

valueOf (String.valueOf method)

```
public valueOf() : String
```

Returns the string.

Availability

Flash Lite 2.0

Returns

String - The value of the string.

Example

The following example creates a new instance of the String object and then shows that the `valueOf` method returns a reference to the *primitive* value, rather than an instance of the object.

```
var str:String = new String("Hello World");
var value:String = str.valueOf();
trace(str instanceof String); // true
trace(value instanceof String); // false
trace(str === value); // false
```

System

```
Object
    |
    +-System
```

```
public class System
extends Object
```

The System class contains properties related to certain operations that take place on the user's computer, such as operations with shared objects and the clipboard. Additional properties and methods are in specific classes within the System package: the capabilities class (see System.capabilities) and the security class (see System.security).

Availability

Flash Lite 2.0

See also

capabilities (System.capabilities), Security (System.security)

Property summary

Modifiers	Property	Description
static	useCodepage : Boolean	A Boolean value that tells Flash Lite player whether to use Unicode or the traditional code page of the operating system running the player to interpret external text files.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Event summary

Event	Description
onStatus = function(infoObject:Object) {}	Event handler: provides a super event handler for certain objects.

Method summary

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

onStatus (System.onStatus handler)

```
onStatus = function(infoObject:Object) {}
```

Event handler: provides a super event handler for certain objects.

The SharedObject class provides an `onStatus()` event handler that uses an information object for providing information, status, or error messages. To respond to this event handler, you must create a function to process the information object, and you must know the format and contents of the returned information object.

In addition to the `SharedObject.onStatus()` method, Flash also provides a super function called `System.onStatus()`, which serves as a secondary error message handler. If an instance of the SharedObject class passes an information object with a level property of "error", but you did not define an `onStatus()` function for that particular instance, then Flash uses the function you define for `System.onStatus()` instead.

Availability

Flash Lite 2.0

Parameters

infoObject : Object - A parameter defined according to the status message.

Example

The following example shows how to create a `System.onStatus()` function to process information objects when a class-specific `onStatus()` function does not exist:

```
// Create generic function
System.onStatus = function(genericError:Object){
    // Your script would do something more meaningful here
    trace("An error has occurred. Please try again.");
}
```

See also

onStatus (SharedObject.onStatus handler)

useCodepage (System.useCodepage property)

```
public static useCodepage : Boolean
```

A Boolean value that tells Flash Lite player whether to use Unicode or the traditional code page of the operating system running the player to interpret external text files. The default value of `System.useCodepage` is false.

- When the property is set to false, Flash Lite player interprets external text files as Unicode. (These files must be encoded as Unicode when you save them.)
- When the property is set to true, Flash Lite player interprets external text files using the traditional code page of the operating system running the player.

Text that you load as an external file (using the `loadVariables()` or `getURL()` statements, or the LoadVars class or XML class) must be encoded as Unicode when you save the text file in order for Flash Lite player to recognize it as Unicode. To encode external files as Unicode, save the files in an application that supports Unicode, such as Notepad on Windows 2000.

If you load external text files that are not Unicode-encoded, you should set `System.useCodepage` to true. Add the following code as the first line of code in the first frame of the SWF file that is loading the data:

```
System.useCodepage = true;
```

When this code is present, Flash Lite player interprets external text using the traditional code page of the operating system running Flash Lite player. This is generally CP1252 for an English Windows operating system and Shift-JIS for a Japanese operating system. If you set `System.useCodepage` to true, Flash Player 6 and later treat text as Flash Player 5 does. (Flash Player 5 treated all text as if it were in the traditional code page of the operating system running the player.)

If you set `System.useCodepage` to true, remember that the traditional code page of the operating system running the player must include the characters used in your external text file in order for the text to display. For example, if you load an external text file that contains Chinese characters, those characters cannot display on a system that uses the CP1252 code page because that code page does not include Chinese characters.

To ensure that users on all platforms can view external text files used in your SWF files, you should encode all external text files as Unicode and leave `System.useCodepage` set to false by default. This way, Flash Player 6 and later interprets the text as Unicode.

Availability
Flash Lite 2.0

TextField

```
Object
    |
    +-TextField

public dynamic class TextField
extends Object
```

The TextField class is used to create areas for text display and input. All dynamic and input text fields are instances of the TextField class. You can give a text field an instance name in the Property inspector and use the methods and properties of the TextField class to manipulate it with ActionScript. TextField instance names are displayed in the Movie Explorer and in the Insert Target Path dialog box in the Actions panel.

To create a text field dynamically, call `MovieClip.createTextField()`.

The methods of the TextField class let you set, select, and manipulate text in a dynamic or input text field that you create during authoring or at runtime.

Availability

Flash Lite 2.0

See also

`Object`, `createTextField (MovieClip.createTextField method)`

Property summary

Modifiers	Property	Description
	_alpha : Number	Sets or retrieves the alpha transparency value of the text field.
	autoSize : Object	Controls automatic sizing and alignment of text fields.
	background : Boolean	Specifies if the text field has a background fill.
	backgroundColor : Number	The color of the text field background.
	border : Boolean	Specifies if the text field has a border.
	borderColor : Number	The color of the text field border.
	bottomScroll : Number [read-only]	An integer (one-based index) that indicates the bottommost line that is currently visible the text field.
	condenseWhite : Boolean	A Boolean value that specifies whether extra white space (spaces, line breaks, and so on) in an HTML text field should be removed when the field is rendered in a browser.
	embedFonts : Boolean	A Boolean value that specifies whether to render the text using embedded font outlines.
	_height : Number	The height of the text field in pixels.
	_highquality : Number	Deprecated since Flash Player 7. This property was deprecated in favor of `TextField._quality`. Specifies the level of anti-aliasing applied to the current SWF file.
	hscroll : Number	Indicates the current horizontal scrolling position.
	html : Boolean	A flag that indicates whether the text field contains an HTML representation.
	htmlText : String	If the text field is an HTML text field, this property contains the HTML representation of the text field's contents.
	length : Number [read-only]	Indicates the number of characters in a text field.
	maxChars : Number	Indicates the maximum number of characters that the text field can contain.
	maxhscroll : Number [read-only]	Indicates the maximum value of `TextField.hscroll`.

Modifiers	Property	Description
	maxscroll : Number [read-only]	Indicates the maximum value of `TextField.scroll`.
	multiline : Boolean	Indicates whether the text field is a multiline text field.
	_name : String	The instance name of the text field.
	_parent : MovieClip	A reference to the movie clip or object that contains the current text field or object.
	password : Boolean	Specifies whether the text field is a password text field.
	_quality : String	Property (global); sets or retrieves the rendering quality used for a SWF file.
	_rotation : Number	The rotation of the text field, in degrees, from its original orientation.
	scroll : Number	Defines the vertical position of text in a text field.
	selectable : Boolean	A Boolean value that indicates whether the text field is selectable.
	_soundbuftime : Number	Specifies the number of seconds a sound prebuffers before it starts to stream.
	tabEnabled : Boolean	Specifies whether the text field is included in automatic tab ordering.
	tabIndex : Number	Lets you customize the tab ordering of objects in a SWF file.
	_target : String [read-only]	The target path of the text field instance.
	text : String	Indicates the current text in the text field.
	textColor : Number	Indicates the color of the text in a text field.
	textHeight : Number	Indicates the height of the text.
	textWidth : Number	Indicates the width of the text.
	type : String	Specifies the type of text field.
	_url : String [read-only]	Retrieves the URL of the SWF file that created the text field.
	variable : String	The name of the variable to which the text field is associated.
	_visible : Boolean	A Boolean value that indicates whether the text field `my_txt` is visible.
	_width : Number	The width of the text field, in pixels.
	wordWrap : Boolean	A Boolean value that indicates if the text field has word wrap.
	_x : Number	An integer that sets the *x* coordinate of a text field relative to the local coordinates of the parent movie clip.
	_xmouse : Number [read-only]	Returns the *x* coordinate of the mouse position relative to the text field.
	_xscale : Number	Determines the horizontal scale of the text field as applied from the registration point of the text field, expressed as a percentage.

Modifiers	Property	Description
	_y : Number	The y coordinate of a text field relative to the local coordinates of the parent movie clip.
	_ymouse : Number [read-only]	Indicates the y coordinate of the mouse position relative to the text field.
	_yscale : Number	The vertical scale of the text field as applied from the registration point of the text field, expressed as a percentage.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Event summary

Event	Description
onChanged = function(changedField:TextField) {}	Event handler/listener; invoked when the content of a text field changes.
onKillFocus = function(newFocus:Object) {}	Invoked when a text field loses keyboard focus.
onScroller = function(scrolledField:TextField) {}	Event handler/listener; invoked when one of the text field scroll properties changes.
onSetFocus = function(oldFocus:Object) {}	Invoked when a text field receives keyboard focus.

Method summary

Modifiers	Signature	Description
	addListener(listener:Object) : Boolean	Registers an object to receive TextField event notifications.
	getDepth() : Number	Returns the depth of a text field.
	getNewTextFormat() : TextFormat	Returns a TextFormat object containing a copy of the text field's text format object.
	getTextFormat([beginIndex:Number], [endIndex:Number]) : TextFormat	Returns a TextFormat object for a character, for a range of characters, or for an entire TextField object.
	removeListener(listener:Object) : Boolean	Removes a listener object previously registered to a text field instance with `TextField.addListener()`.
	removeTextField() : Void	Removes the text field.
	replaceSel(newText:String) : Void	Replaces the current selection with the contents of the `newText` parameter.

Modifiers	Signature	Description
	replaceText(beginIndex:Number, endIndex:Number, newText:String) : Void	Replaces a range of characters, specified by the beginIndex and endIndex parameters, in the specified text field with the contents of the newText parameter.
	setNewTextFormat(tf:TextFormat) : Void	Sets the default new text format of a text field.
	setTextFormat([beginIndex:Number], [endIndex:Number], textFormat:TextFormat) : Void	Applies the text formatting specified by the textFormat parameter to some or all of the text in a text field.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

addListener (TextField.addListener method)

```
public addListener(listener:Object) : Boolean
```

Registers an object to receive TextField event notifications. The object will receive event notifications whenever the onChanged and onScroller event handlers have been invoked. When a text field changes or is scrolled, the TextField.onChanged and TextField.onScroller event handlers are invoked, followed by the onChanged and onScroller event handlers of any objects registered as listeners. Multiple objects can be registered as listeners.

To remove a listener object from a text field, call TextField.removeListener().

A reference to the text field instance is passed as a parameter to the onScroller and onChanged handlers by the event source. You can capture this data by putting a parameter in the event handler method. For example, the following code uses txt as the parameter that is passed to the onScroller event handler. The parameter is then used in a trace statement to send the instance name of the text field to the Output panel.The parameter is then used in a trace() method to write the instance name of the text field to the log file.

```
my_txt.onScroller = function(textfield_txt:TextField) {
    trace(textfield_txt._name+" scrolled");
};
```

Availability

Flash Lite 2.0

Parameters

listener: Object - An object with an onChanged or onScroller event handler.

Returns

Boolean -

Example

The following example defines an `onChanged` handler for the input text field `my_txt`. It then defines a new listener object, `txtListener`, and defines an `onChanged` handler for that object. This handler will be invoked when the text field `my_txt` is changed. The final line of code calls `TextField.addListener` to register the listener object `txtListener` with the text field `my_txt` so that it will be notified when `my_txt` changes.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
my_txt.border = true;
my_txt.type = "input";

my_txt.onChanged = function(textfield_(xt:TextField) {
    trace(textfield_txt._name+" changed");
};
var txtListener:Object = new Object();
txtListener.onChanged = function(textfield_txt:TextField) {
    trace(textfield_txt._name+" changed and notified myListener");
};
my_txt.addListener(txtListener);
```

See also

onChanged (TextField.onChanged handler), onScroller (TextField.onScroller handler), removeListener (TextField.removeListener method)

_alpha (TextField._alpha property)

```
public _alpha : Number
```

Sets or retrieves the alpha transparency value of the text field. Valid values are 0 (fully transparent) to 100 (fully opaque). The default value is 100. Transparency values are not supported for text fields that use device fonts. You must use embedded fonts to use the `_alpha` transparency property with a text field.

Note: This property is not supported for Arabic, Hebrew, and Thai.

Availability

Flash Lite 2.0

Example

The following code sets the `_alpha` property of a text field named `my_txt` to 20%. Create a new font symbol in the library by selecting New Font from the Library options menu. Then set the linkage of the font to `my font`. Set the linkage for a font symbol to `my font`. Add the following ActionScript to your FLA or ActionScript file:

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";
// where 'my font' is the linkage name of a font in the Library
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
my_txt.border = true;
my_txt.embedFonts = true;
my_txt.text = "Hello World";
my_txt.setTextFormat(my_fmt);
my_txt._alpha = 20;
```

See also

_alpha (Button._alpha property), _alpha (MovieClip._alpha property)

autoSize (TextField.autoSize property)

```
public autoSize : Object
```

Controls automatic sizing and alignment of text fields. Acceptable values for autoSize are `"none"` (the default), `"left"`, `"right"`, and `"center"`. When you set the `autoSize` property, `true` is a synonym for `"left"` and `false` is a synonym for `"none"`.

The values of `autoSize` and `TextField.wordWrap` determine whether a text field expands or contracts to the left side, right side, or bottom side. The default value for each of these properties is `false`.

If `autoSize` is set to `"none"` (the default) or `false`, then no resizing will occur.

If `autoSize` is set to `"left"` or `true`, then the text is treated as left-justified text, meaning the left side of the text field will remain fixed and any resizing of a single line text field will be on the right side. If the text includes a line break (for example, `"\n" or "\r")`, then the bottom side will also be resized to fit the next line of text. If `wordWrap` is also set to `true`, then only the bottom side of the text field will be resized and the right side will remain fixed.

If `autoSize` is set to `"right"`, then the text is treated as right-justified text, meaning the right side of the text field will remain fixed and any resizing of a single line text field will be on the left side. If the text includes a line break (for example, `"\n" or "\r")`, then the bottom side will also be resized to fit the next line of text. If `wordWrap` is also set to `true`, then only the bottom side of the text field will be resized and the left side will remain fixed.

If `autoSize` is set to `"center"`, then the text is treated as center-justified text, meaning any resizing of a single line text field will be equally distributed to both the right and left sides. If the text includes a line break (for example, `"\n" or "\r")`, then the bottom side will also be resized to fit the next line of text. If `wordWrap` is also set to `true`, then only the bottom side of the text field will be resized and the left and right sides will remain fixed.

Availability

Flash Lite 2.0

Example

You can use the following code and enter different values for `autoSize` to see how the field resizes when these values change. A mouse click while the SWF file is playing will replace each text field's `"short text"` string with longer text using several different settings for `autoSize`.

```
this.createTextField("left_txt", 997, 10, 10, 70, 30);
this.createTextField("center_txt", 998, 10, 50, 70, 30);
this.createTextField("right_txt", 999, 10, 100, 70, 30);
this.createTextField("true_txt", 1000, 10, 150, 70, 30);
this.createTextField("false_txt", 1001, 10, 200, 70, 30);

left_txt.text = "short text";
left_txt.border = true;

center_txt.text = "short text";
center_txt.border = true;

right_txt.text = "short text";
right_txt.border = true;

true_txt.text = "short text";
true_txt.border = true;

false_txt.text = "short text";
false_txt.border = true;

// create a mouse listener object to detect mouse clicks
var myMouseListener:Object = new Object();
// define a function that executes when a user clicks the mouse
myMouseListener.onMouseDown = function() {
    left_txt.autoSize = "left";
    left_txt.text = "This is much longer text";
    center_txt.autoSize = "center";
    center_txt.text = "This is much longer text";
    right_txt.autoSize = "right";
    right_txt.text = "This is much longer text";
    true_txt.autoSize = true;
    true_txt.text = "This is much longer text";
    false_txt.autoSize = false;
    false_txt.text = "This is much longer text";
};
// register the listener object with the Mouse object
Mouse.addListener(myMouseListener);
```

background (TextField.background property)

```
public background : Boolean
```

Specifies if the text field has a background fill. If `true`, the text field has a background fill. If `false`, the text field has no background fill.

Availability

Flash Lite 2.0

Example

The following example creates a text field with a background color that toggles on and off when nearly any key on the keyboard is pressed.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
my_txt.backgroundColor = 0xFF0000;

var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    my_txt.background = !my_txt.background;
};
Key.addListener(keyListener);
```

backgroundColor (TextField.backgroundColor property)

```
public backgroundColor : Number
```

The color of the text field background. Default is `0xFFFFFF` (white). This property may be retrieved or set, even if there currently is no background, but the color is only visible if the text field has a border.

Availability

Flash Lite 2.0

Example

See the example for `TextField.background`.

See also

`background (TextField.background property)`

border (TextField.border property)

```
public border : Boolean
```

Specifies if the text field has a border. If `true`, the text field has a border. If `false`, the text field has no border.

Availability

Flash Lite 2.0

Example

The following example creates a text field called `my_txt`, sets the border property to `true`, and displays some text in the field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
```

borderColor (TextField.borderColor property)

```
public borderColor : Number
```

The color of the text field border. The default is `0x000000` (black). This property may be retrieved or set, even if there is currently no border.

Availability

Flash Lite 2.0

Example

The following example creates a text field called `my_txt`, sets the border property to `true`, and displays some text in the field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 100);
my_txt.border = true;
my_txt.borderColor = 0x00FF00;
my_txt.text = "Lorum ipsum";
```

See also

`border (TextField.border property)`

bottomScroll (TextField.bottomScroll property)

```
public bottomScroll : Number [read-only]
```

An integer (one-based index) that indicates the bottommost line that is currently visible the text field. Think of the text field as a window onto a block of text. The property `TextField.scroll` is the one-based index of the topmost visible line in the window.

All the text between lines `TextField.scroll` and `TextField.bottomScroll` is currently visible in the text field.

Availability

Flash Lite 2.0

Example

The following example creates a text field and fills it with text. Create a button with the instance name `my_btn`, and when you click it, the `scroll` and `bottomScroll` properties for the text field display in the `comment_txt` field.

```
this.createTextField("comment_txt", this.getNextHighestDepth(), 0, 0, 160, 120);
comment_txt.html = true;
comment_txt.selectable = true;
comment_txt.multiline = true;
comment_txt.wordWrap = true;
comment_txt.htmlText = "<b>What is hexadecimal?</b><br>"
    + "The hexadecimal color system uses six digits to represent color values. "
    + "Each digit has sixteen possible values or characters. The characters range"
    + " from 0 to 9 and then A to F. Black is represented by (#000000) and white, "
    + "at the (pposite end of the color system, is (#FFFFFF).";
my_btn.onRelease = function() {
    trace("scroll: "+comment_txt.scroll);
    trace("bottomScroll: "+comment_txt.bottomScroll);
};
```

condenseWhite (TextField.condenseWhite property)

```
public condenseWhite : Boolean
```

A Boolean value that specifies whether extra white space (spaces, line breaks, and so on) in an HTML text field should be removed when the field is rendered in a browser. The default value is `false`.

If you set this value to `true`, you must use standard HTML commands such as `<BR>` and `<P>` to place line breaks in the text field.

If the text field's `.html` is `false`, this property is ignored.

Availability
Flash Lite 2.0

Example
The following example creates two text fields, called `first_txt` and `second_txt`. The white space is removed from the second text field. Add the following ActionScript to your FLA or ActionScript file:

```
var my_str:String = "Hello\tWorld\nHow are you?\t\t\tEnd";

this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
first_txt.html = true;
first_txt.multiline = true;
first_txt.wordWrap = true;
first_txt.condenseWhite = false;
first_txt.border = true;
first_txt.htmlText = my_str;

this.createTextField("second_txt", this.getNextHighestDepth(), 180, 10, 160, 120);
second_txt.html = true;
second_txt.multiline = true;
second_txt.wordWrap = true;
second_txt.condenseWhite = true;
second_txt.border = true;
second_txt.htmlText = my_str;
```

See also
`html (TextField.html property)`

embedFonts (TextField.embedFonts property)

```
public embedFonts : Boolean
```

A Boolean value that specifies whether to render the text using embedded font outlines. If the value is `true`, Flash Lite renders the text field using embedded font outlines. If the value is `false`, Flash Lite renders the text field using device fonts.

If you set `embedFonts` to `true` for a text field, you must specify a font for that text using the `font` property of a TextFormat object applied to the text field. If the specified font does *not* exist in the library (with the corresponding linkage identifier), the text is not displayed.

Note: This property is not supported for Arabic, Hebrew, and Thai.

Availability
Flash Lite 2.0

Example
In this example, you need to create a dynamic text field called `my_txt`, and then use the following ActionScript to embed fonts and rotate the text field. The string `my font` refers to a font symbol in the library, with the linkage identifier name `my font`. The example assumes that you have a font symbol in the library called `my font`, with linkage properties set as follows: the identifier set to `my font` and Export for ActionScript and Export in First Frame selected.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";

this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

getDepth (TextField.getDepth method)

```
public getDepth() : Number
```

Returns the depth of a text field.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

The following example demonstrates text fields residing at different depths. Create a dynamic text field on the Stage. Add the following ActionScript to your FLA or ActionScript file, which dynamically creates two text fields at runtime and outputs their depths.

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
for (var prop in this) {
    if (this[prop] instanceof TextField) {
    var this_txt:TextField = this[prop];
    trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
    }
}
```

getNewTextFormat (TextField.getNewTextFormat method)

```
public getNewTextFormat() : TextFormat
```

Returns a TextFormat object containing a copy of the text field's text format object. The text format object is the format that newly inserted text, such as text entered by a user, receives. When `getNewTextFormat()` is invoked, the TextFormat object returned has all of its properties defined. No property is `null`.

Availability

Flash Lite 2.0

Returns

TextFormat - A TextFormat object.

Example

The following example displays the specified text field's (`my_txt`) TextFormat object.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
var my_fmt:TextFormat = my_txt.getNewTextFormat();
trace("TextFormat has the following properties:");
for (var prop in my_fmt) {
    trace(prop+": "+my_fmt[prop]);
}
```

getTextFormat (TextField.getTextFormat method)

```
public getTextFormat([beginIndex:Number], [endIndex:Number]) : TextFormat
```

Returns a TextFormat object for a character, for a range of characters, or for an entire TextField object.

- **Usage 1:** `my_textField.getTextFormat()`

  Returns a TextFormat object containing formatting information for all text in a text field. Only properties that are common to all text in the text field are set in the resulting TextFormat object. Any property which is *mixed*, meaning that it has different values at different points in the text, has a value of `null`.

- **Usage 2:** `my_textField.getTextFormat(beginIndex:Number)`

  Returns a TextFormat object containing a copy of the text field's text format at the `beginIndex` position.

- **Usage 3:** `my_(extField.getTextFormat(beginIndex:Number,endIndex:Number)`

  Returns a TextFormat object containing formatting information for the span of text from `beginIndex` to `endIndex`. Only properties that are common to all of the text in the specified range is set in the resulting TextFormat object. Any property that is mixed (it has different values at different points in the range) has its value set to null.

Availability

Flash Lite 2.0

Parameters

beginIndex: Number [optional] - An integer that specifies a character in a string. If you do not specify `beginIndex` and `endIndex`, the TextFormat object returned is for the entire TextField.

endIndex: Number [optional] - An integer that specifies the end position of a span of text. If you specify `beginIndex` but do not specify `endIndex`, the TextFormat returned is for the single character specified by `beginIndex`.

Returns

TextFormat - An object.

Example

The following ActionScript traces all of the formatting information for a text field that is created at runtime.

```
this.createTextField("dyn_txt", this.getNextHighestDepth(), 0, 0, 100, 200);
dyn_txt.text = "Frank";
dyn_txt.setTextFormat(new TextFormat());
var my_fmt:TextFormat = dyn_txt.getTextFormat();
for (var prop in my_fmt) {
    trace(prop+": "+my_fmt[prop]);
}
```

See also

getNewTextFormat (TextField.getNewTextFormat method), setNewTextFormat (TextField.setNewTextFormat method) setTextFormat (TextField.setTextFormat method)

_height (TextField._height property)

```
public _height : Number
```

The height of the text field in pixels.

Availability

Flash Lite 2.0

Example

The following code example sets the height and width of a text field:

```
my_txt._width = 200;
my_txt._height = 200;
```

_highquality (TextField._highquality property)

```
public _highquality : Number
```

Deprecated since Flash Player 7. This property was deprecated in favor of `TextField._quality`.

Specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this smooths bitmaps if the SWF file does not contain animation and is the default value. Specify 0 (low quality) to prevent anti-aliasing.

Availability

Flash Lite 2.0

See also

_quality (TextField._quality property)

hscroll (TextField.hscroll property)

```
public hscroll : Number
```

Indicates the current horizontal scrolling position. If the `hscroll` property is 0, the text is not horizontally scrolled.

The units of horizontal scrolling are pixels, while the units of vertical scrolling are lines. Horizontal scrolling is measured in pixels because most fonts that are typically used are proportionally spaced, and therefore the characters can have different widths. Flash performs vertical scrolling by line because users usually want to see a line of text in its entirety, as opposed to seeing a partial line. Even if there are multiple fonts on a line, the height of the line adjusts to fit the largest font in use.

Note: The `hscroll` property is zero-based, not one-based like the vertical scrolling property `TextField.scroll`.

Availability

Flash Lite 2.0

Example

The following example scrolls the `my_txt` text field horizontally using two buttons called `scrollLeft_btn` and `scrollRight_btn`. The amount of scroll appears in a text field called `scroll_txt`. Add the following ActionScript to your FLA or ActionScript file:

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 160, 22);
my_txt.border = true;
my_txt.multiline = false;
my_txt.wordWrap = false;
my_txt.text = "Lorem ipsum dolor sit amet, consectetuer adipiscing...";

scrollLeft_btn.onRelease = function() {
    my_txt.hscroll -= 10;
    scroll_txt.text = my_txt.hscroll+" of "+my_txt.maxhscroll;
};
scrollRight_btn.onRelease = function() {
    my_txt.hscroll += 10;
    scroll_txt.text = my_txt.hscroll+" of "+my_txt.maxhscroll;
};
```

See also

maxhscroll (TextField.maxhscroll property), scroll (TextField.scroll property)

html (TextField.html property)

```
public html : Boolean
```

A flag that indicates whether the text field contains an HTML representation. If the `html` property is `true`, the text field is an HTML text field. If `html` is `false`, the text field is a non-HTML text field.

Availability

Flash Lite 2.0

Example

The following example creates a text field that sets the `html` property to `true`. HTML-formatted text appears in the text field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "<b> this is bold text </b>";
```

See also

htmlText (TextField.htmlText property)

htmlText (TextField.htmlText property)

```
public htmlText : String
```

If the text field is an HTML text field, this property contains the HTML representation of the text field's contents. If the text field is not an HTML text field, it behaves identically to the `text` property. You can indicate that a text field is an HTML text field in the Property inspector, or by setting the text field's `html` property to `true`.

Availability
Flash Lite 2.0

Example
The following example creates a text field that sets the `html` property to `true`. HTML-formatted text appears in the text field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "< this is bold text >";
```

See also
`html (TextField.html property)`

length (TextField.length property)

```
public length : Number [read-only]
```

Indicates the number of characters in a text field. This property returns the same value as `text.length`, but is faster. A character such as tab (`\t`) counts as one character.

Availability
Flash Lite 2.0

Example
The following example outputs the number of characters in the `date_txt` text field, which displays the current date.

```
var today:Date = new Date();
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
date_txt.autoSize = true;
date_txt.text = today.toString();
trace(date_txt.length);
```

maxChars (TextField.maxChars property)

```
public maxChars : Number
```

Indicates the maximum number of characters that the text field can contain. A script may insert more text than `maxChars` allows; the `maxChars` property indicates only how much text a user can enter. If the value of this property is `null`, there is no limit on the amount of text a user can enter.

Availability
Flash Lite 2.0

Example
The following example creates a text field called `age_txt` that only lets users enter up to two numbers in the field.

```
this.createTextField("age_txt", this.getNextHighestDepth(), 10, 10, 30, 22);
age_txt.type = "input";
age_txt.border = true;
age_txt.maxChars = 2;
```

maxhscroll (TextField.maxhscroll property)

```
public maxhscroll : Number [read-only]
```

Indicates the maximum value of `TextField.hscroll`.

Availability

Flash Lite 2.0

Example

See the example for `TextField.hscroll`.

maxscroll (TextField.maxscroll property)

```
public maxscroll : Number [read-only]
```

Indicates the maximum value of `TextField.scroll`.

Availability

Flash Lite 2.0

Example

The following example sets the maximum value for the scrolling text field `my_txt`. Create two buttons, `scrollUp_btn` and `scrollDown_btn`, to scroll the text field. Add the following ActionScript to your FLA or ActionScript file.

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy
nibh "
            + "euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

multiline (TextField.multiline property)

```
public multiline : Boolean
```

Indicates whether the text field is a multiline text field. If the value is `true`, the text field is multiline; if the value is `false`, the text field is a single-line text field.

Availability

Flash Lite 2.0

Example

The following example creates a multiline text field called `myText`.

```
this.createTextField("myText", this.getNextHighestDepth(), 10, 30, 110, 100);
myText.text = "Flash is an authoring tool that designers and developers use to create
presentations,
applications, and other content that enables user interaction.";
myText.border = true;
myText.wordWrap = true;
myText.multiline = true;
```

_name (TextField._name property)

```
public _name : String
```

The instance name of the text field.

Availability

Flash Lite 2.0

Example

The following example demonstrates text fields residing at different depths. Create a dynamic text field on the Stage. Add the following ActionScript to your FLA or ActionScript file, which dynamically creates two text fields at runtime and displays their depths in the Output panel.

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
for (var prop in this) {
    if (this[prop] instanceof TextField) {
    var this_txt:TextField = this[prop];
    trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
    }
}
```

When you test the document, the instance name and depth is displayed in the Output panel.When you test the document, the instance name and depth writes to the log file.

onChanged (TextField.onChanged handler)

```
onChanged = function(changedField:TextField) {}
```

Invoked when the content of a text field changes. By default, it is undefined. You can define it in a script.

A reference to the text field instance is passed as a parameter to the `onChanged` handler. You can capture this data by putting a parameter in the event handler method. For example, the following code uses `textfield_txt` as the parameter that is passed to the `onChanged` event handler. The parameter is then used in a `trace()` statement to send the instance name of the text field to the Output panel:

```
this.createTextField("myInputText_txt", 99, 10, 10, 300, 20);
myInputText_txt.border = true;
myInputText_txt.type = "input";

myInputText_txt.onChanged = function(textfield_txt:TextField) {
trace("the value of "+textfield_txt._name+" was changed. New value is: "+textfield_txt.text);
};
```

The `onChanged` handler is called only when the change results from user interaction; for example, when the user is typing something on the keyboard, changing something in the text field using the mouse, or selecting a menu item. Programmatic changes to the text field do not trigger the `onChanged` event because the code recognizes changes that are made to the text field.

Availability
Flash Lite 2.0

Parameters
changedField : TextField - The field triggering the event.

See also
`getNewTextFormat (TextField.getNewTextFormat method)`, `setNewTextFormat (TextField.setNewTextFormat method)`

onKillFocus (TextField.onKillFocus handler)

```
onKillFocus = function(newFocus:Object) {}
```

Invoked when a text field loses keyboard focus. The `onKillFocus` method receives one parameter, `newFocus`, which is an object representing the new object receiving the focus. If no object receives the focus, `newFocus` contains the value `null`.

Availability
Flash Lite 2.0

Parameters
newFocus : Object - The object that is receiving the focus.

Example
The following example creates two text fields called `first_txt` and `second_txt`. When you give focus to a text field, information about the text field with current focus and the text field that lost focus is displayed in the Output panel.

```
this.createTextField("first_txt", 1, 10, 10, 300, 20);
first_txt.border = true;
first_txt.type = "input";
this.createTextField("second_txt", 2, 10, 40, 300, 20);
second_txt.border = true;
second_txt.type = "input";
first_txt.onKillFocus = function(newFocus:Object) {
    trace(this._name+" lost focus. New focus changed to: "+newFocus._name);
};
first_txt.onSetFocus = function(oldFocus:Object) {
    trace(this._name+" gained focus. Old focus changed from: "+oldFocus._name);
}
```

See also
`onSetFocus (TextField.onSetFocus handler)`

onScroller (TextField.onScroller handler)

```
onScroller = function(scrolledField:TextField) {}
```

Invoked when one of the text field scroll properties changes.

A reference to the text field instance is passed as a parameter to the `onScroller` handler. You can capture this data by putting a parameter in the event handler method. For example, the following code uses `my_txt` as the parameter that is passed to the `onScroller` event handler. The parameter is then used in a `trace()` statement to send the instance name of the text field to the Output panel.

```
myTextField.onScroller = function (my_txt:TextField) {
    trace (my_txt._name + " scrolled");
};
```

The `TextField.onScroller` event handler is commonly used to implement scroll bars. Scroll bars typically have a thumb or other indicator that shows the current horizontal or vertical scrolling position in a text field. Text fields can be navigated using the mouse and keyboard, which causes the scroll position to change. The scroll bar code needs to be notified if the scroll position changes because of such user interaction. This is what `TextField.onScroller` is used for.

`onScroller` is called whether the scroll position changed because of a user's interaction with the text field, or due to programmatic changes. The `onChanged` handler fires only if a user interaction causes the change. These two options are necessary because often one piece of code changes the scrolling position, while the scroll bar code is unrelated and won't know that the scroll position changed without being notified.

Availability
Flash Lite 2.0

Parameters
scrolledField : TextField - A reference to the TextField object whose scroll position was changed.

Example
The following example creates a text field called `my_txt`, and uses two buttons called `scrollUp_btn` and `scrollDown_btn` to scroll the contents of the text field. When the `onScroller` event handler is called, a trace statement is used to display information in the Output panel. Create two buttons with instance names `scrollUp_btn` and `scrollDown_btn`, and add the following ActionScript to your FLA or ActionScript file:

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;

for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam "
        + "nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
};
my_txt.onScroller = function() {
        trace("onScroller called");
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

See also

hscroll (TextField.hscroll property), maxhscroll (TextField.maxhscroll property), maxscroll (TextField.maxscroll property) scroll (TextField.scroll property)

onSetFocus (TextField.onSetFocus handler)

```
onSetFocus = function(oldFocus:Object) {}
```

Invoked when a text field receives keyboard focus. The `oldFocus` parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a button to a text field, `oldFocus` contains the button instance. If there is no previously focused object, `oldFocus` contains a null value.

Availability

Flash Lite 2.0

Parameters

oldFocus : Object - The object to lose focus.

Example

See the example for `TextField.onKillFocus`.

See also

onKillFocus (TextField.onKillFocus handler)

_parent (TextField._parent property)

```
public _parent : MovieClip
```

A reference to the movie clip or object that contains the current text field or object. The current object is the one containing the ActionScript code that references `_parent`.

Use `_parent` to specify a relative path to movie clips or objects that are above the current text field. You can use `_parent` to climb up multiple levels in the display list as in the following:

```
_parent._parent._alpha = 20;
```

Availability

Flash Lite 2.0

Example

The following ActionScript creates two text fields and outputs information about the `_parent` of each object. The first text field, `first_txt`, is created on the main Timeline. The second text field, `second_txt`, is created inside the movie clip called `holder_mc`.

```
this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
first_txt.border = true;
trace(first_txt._name+"'s _parent is: "+first_txt._parent);

this.createEmptyMovieClip("holder_mc", this.getNextHighestDepth());
holder_mc.createTextField("second_txt", holder_mc.getNextHighestDepth(), 10, 40, 160, 22);
holder_mc.second_txt.border = true;
trace(holder_mc.second_txt._name+"'s _parent is: "+holder_mc.second_txt._parent);
```

The following information is displayed in the Output panel:

```
first_txt'( _parent is: _level0
second_txt's _parent is: _level0.holder_mc
```

See also

_parent (Button._parent property), _parent (MovieClip._parent property), _root property

password (TextField.password property)

```
public password : Boolean
```

Specifies whether the text field is a password text field. If the value of password is `true`, the text field is a password text field: once the user completes entering the password and clicks OK, the text field hides the input characters using asterisks instead of the actual characters. If `false`, the text field is not a password text field. When password mode is enabled, the *Cut* and *Copy* commands and their corresponding keyboard accelerators will not function. This security mechanism prevents an unscrupulous user from using the shortcuts to discover a password on an unattended computer.

Availability

Flash Lite 2.0

Example

The following example creates two text fields: `username_txt` and `password_txt`. Text is entered into both text fields; however, `password_txt` has the password property set to `true`. After the user clicks OK to complete the password entry, the characters display as asterisks instead of as characters in the `password_txt` field.

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
username_txt.border = true;
username_txt.type = "input";
username_txt.maxChars = 16;
username_txt.text = "hello";

this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
password_txt.border = true;
password_txt.type = "input";
password_txt.maxChars = 16;
password_tx(.password = true;
password_txt.text = "world";
```

_quality (TextField._quality property)

```
public _quality : String
```

Property (global); sets or retrieves the rendering quality used for a SWF file. Device fonts are always aliased and, therefore, are unaffected by the `_quality` property.

Note: Although you can specify this property for a TextField object, it is actually a global property, and you can specify its value simply as `_quality`. For more information, see the `_quality` property.

The `_quality` property can be set to the following values:

- "`LOW`" Low rendering quality. Graphics are not anti-aliased, and bitmaps are not smoothed.
- "`MEDIUM`" Medium rendering quality. Graphics are anti-aliased using a 2 x 2 pixel grid, but bitmaps are not smoothed. The quality is suitable for movies that do not contain text.

- "HIGH" High rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are smoothed if the movie is static. This is the default rendering quality used by Flash.
- "BEST" Very high rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid and bitmaps are always smoothed.

Note: This property is not supported for Arabic, Hebrew, and Thai.

Availability
Flash Lite 2.0

Example
The following example sets the rendering quality to LOW:

```
my_txt._quality = "LOW";
```

See also
_quality property

removeListener (TextField.removeListener method)

```
public removeListener(listener:Object) : Boolean
```

Removes a listener object previously registered to a text field instance with TextField.addListener().

Availability
Flash Lite 2.0

Parameters
listener : Object - The object that will no longer receive notifications from TextField.onChanged or TextField.onScroller.

Returns
Boolean - If listener was successfully removed, the method returns a true value. If listener was not successfully removed (for example, if listener was not on the TextField object's listener list), the method returns a value of false.

Example
The following example creates an input text field called my_txt. When the user types into the field, information about the number of characters in the text field is displayed in the Output panel. When the user types into the field, information about the number of characters in the text field writes to the log file. If the user clicks the removeListener_btn instance, then the listener is removed and the information is no longer displayed. If the user clicks the removeListener_btn instance, then the listener is removed and the information no longer writes to the log file.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
my_txt.border = true;
my_txt.type = "input";

var txtListener:Object = new Object();
txtListener.onChanged = function(textfield_txt:TextField) {
    trace(textfield_txt+" changed. Current length is: "+textfield_txt.length);
};
my_txt.addListener(txtListener);

removeListener_btn.onRelease = function() {
    trace("Removing listener...");
    if (!my_txt.removeListener(txtListener)) {
    trace("Error! Unable to remove listener");
    }
};
```

(

removeTextField (TextField.removeTextField method)

```
public removeTextField() : Void
```

Removes the text field. This operation can only be performed on a text field that was created with `MovieClip.createTextField()`. When you call this method, the text field is removed. This method is similar to `MovieClip.removeMovieClip()`.

Availability

Flash Lite 2.0

Example

The following example creates a text field that you can remove from the Stage when you click the `remove_btn` instance. Create a button and call it `remove_btn`, and then add the following ActionScript to your FLA or ActionScript file.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 300, 22);
my_txt.text = new Date().toString();
my_txt.border = true;

remove_btn.onRelease = function() {
    my_txt.removeTextField();
};
```

replaceSel (TextField.replaceSel method)

```
public replaceSel(newText:String) : Void
```

Replaces the current selection with the contents of the `newText` parameter. The text is inserted at the position of the current selection, using the current default character format and default paragraph format. The text is not treated as HTML, even if the text field is an HTML text field.

You can use the `replaceSel()` method to insert and delete text without disrupting the character and paragraph formatting of the rest of the text.

Note: You must use the `Selection.setFocus()` method to focus the field before you call the `replaceSel()` method.

Availability
Flash Lite 2.0

Parameters
newText : String - A string.

Example
The following example code creates a multiline text field with text on the Stage. When you select some text and then right-click or Control-click over the text field, you can select `Enter current date` from the context menu. This selection calls a function that replaces the selected text with the current date.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.wordWrap = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.text = "Select some sample text from the text field and then right-click/control click "
        + "and select 'Enter current date' from the context menu to replace the "
        + "currently selected text with the current date.";

var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Enter current date", enterDate));
function enterDate(obj:Object, menuItem:ContextMenuItem) {
    var today_str:String = new Date().toString();
    var date_str:String = today_str.split(" ", 3).join(" ");
    my_txt.replaceSel(date_str);
}
my_txt.menu = my_cm;
```

See also
setFocus (Selection.setFocus method)

replaceText (TextField.replaceText method)

```
public replaceText(beginIndex:Number, endIndex:Number, newText:String) : Void
```

Replaces a range of characters, specified by the `beginIndex` and `endIndex` parameters, in the specified text field with the contents of the `newText` parameter.

Availability
Flash Lite 2.0

Parameters
beginIndex : Number - The start index value for the replacement range.

endIndex : Number - The end index value for the replacement range.

newText : String - The text to use to replace the specified range of characters.

Example

The following example creates a text field called `my_txt` and assigns the text `dog@house.net` to the field. The `indexOf()` method is used to find the first occurrence of the specified symbol (`@`). If the symbol is found, the specified text (between the index of 0 and the symbol) replaces with the string `bird`. If the symbol is not found, an error message is displayed in the Output panel. If the symbol is not found, an error message writes to the log file.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 22);
my_txt.autoSize = true;
my_txt.text = "dog@house.net";

var symbol:String = "@";
var symbolPos:Number = my_txt.text.indexOf(symbol);
if (symbolPos>-1) {
    my_txt.replaceText(0, symbolPos, "bird");
} else {
    trace("symbol '"+symbol+"' not found.");
}
```

_rotation (TextField._rotation property)

```
public _rotation : Number
```

The rotation of the text field, in degrees, from its original orientation. Values from 0 to 180 represent clockwise rotation; values from 0 to -180 represent counterclockwise rotation. Values outside this range are added to or subtracted from 360 to obtain a value within the range. For example, the statement `my_txt._rotation = 450` is the same as `my_txt._rotation = 90`.

Rotation values are not supported for text fields that use device fonts. You must use embedded fonts to use `_rotation` with a text field.

Note: This property is not supported for Arabic, Hebrew, and Thai.

Availability

Flash Lite 2.0

Example

In this example, you need to create a dynamic text field called `my_txt`, and then use the following ActionScript to embed fonts and rotate the text field. The string `my font` refers to a font symbol in the library, with a linkage identifier of `my font`.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";

this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

Apply additional formatting for the text field using the TextFormat class.

See also

`_rotation (Button._rotation property)`, `_rotation (MovieClip._rotation property)`, `getNewTextFormat (TextField.getNewTextFormat method)`

scroll (TextField.scroll property)

`public scroll : Number`

Defines the vertical position of text in a text field. The scroll property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text fields. This property can be retrieved and modified.

The units of horizontal scrolling are pixels, while the units of vertical scrolling are lines. Horizontal scrolling is measured in pixels because most fonts that are typically used are proportionally spaced. This means that the characters can have different widths. Flash performs vertical scrolling by line because users usually want to see a line of text in its entirety, as opposed to seeing a partial line. Even if there are multiple fonts on a line, the height of the line adjusts to fit the largest font in use.

Availability

Flash Lite 2.0

Example

The following example sets the maximum value for the scrolling text field `my_txt`. Create two buttons, `scrollUp_btn` and `scrollDown_btn`, to scroll the text field. Add the following ActionScript to your FLA or ActionScript file.

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy "
        + "nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

See also

`hscroll (TextField.hscroll property)`, `maxscroll (TextField.maxscroll property)`

selectable (TextField.selectable property)

`public selectable : Boolean`

A Boolean value that indicates whether the text field is selectable. If the value is `true`, the text is selectable. The `selectable` property controls whether a text field is selectable, not whether a text field is editable. A dynamic text field can be selectable even if it is not editable. If a dynamic text field is not selectable, you cannot select its text.

If `selectable` is set to `false`, the text in the text field does not respond to selection commands from the mouse or keyboard, and the text cannot be copied using the Copy command. If selectable is set to `true`, the text in the text field can be selected using the mouse or keyboard. You can select text this way even if the text field is a dynamic text field instead of an input text field. You can also copy the text using the Copy command.

Note: This property is not supported for Arabic, Hebrew, and Thai.

Availability
Flash Lite 2.0

Example
The following example creates a selectable text field that is constantly updated with the current date and time.

```
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
date_txt.autoSize = true;
date_txt.selectable = true;

( var date_interval:Number = setInterval(updateTime, 500, date_txt);
function updateTime(my_txt:TextField) {
    my_txt.text = new Date().toString();
}
```

setNewTextFormat (TextField.setNewTextFormat method)

```
public setNewTextFormat(tf:TextFormat) : Void
```

Sets the default new text format of a text field. The default new text format is the new text format used for newly inserted text such as text entered by a user. When text is inserted, the newly inserted text is assigned the default new text format.

The new default text format is specified by `textFormat`, which is a TextFormat object.

Availability
Flash Lite 2.0

Parameters
tf: TextFormat - A TextFormat object.

Example
In the following example, a new text field (called `my_txt`) is created at runtime and several properties are set. The format of the newly inserted text is applied.

```
var my_fmt:TextForma( = new TextFormat();
my_fmt.bold = true;
my_fmt.font = "Arial";
my_fmt.color = 0xFF9900;

this.createTextField("my_txt", 999, 0, 0, 400, 300);
my_txt.wordWrap = true;
my_txt.multiline = true;
my_txt.border = true;
my_txt.type = "input";
my_txt.setNewTextFormat(my_fmt);
my_txt.text = "Oranges are a good source of vitamin C";
```

See also
getNewTextFormat (TextField.getNewTextFormat method), getTextFormat (TextField.getTextFormat method) setTextFormat (TextField.setTextFormat method)

setTextFormat (TextField.setTextFormat method)

```
public setTextFormat([beginIndex:Number], [endIndex:Number], textFormat:TextFormat) : Void
```

Applies the text formatting specified by the `textFormat` parameter to some or all of the text in a text field. `textFormat` must be a TextFormat object that specifies the text formatting changes desired. Only the non-null properties of `TextFormat` are applied to the text field. Any property of `textFormat` that is set to null will not be applied. By default, all of the properties of a newly created TextFormat object are set to `null`.

There are two types of formatting information in a TextFormat object: character level, and paragraph level formatting. Each character in a text field might have its own character formatting settings, such as font name, font size, bold, and italic.

For paragraphs, the first character of the paragraph is examined for the paragraph formatting settings for the entire paragraph. Examples of paragraph formatting settings are left margin, right margin, and indentation.

The `setTextFormat()` method changes the text formatting applied to an individual character, to a range of characters, or to the entire body of text in a text field:

- **Usage 1:** `my_textField.setTextFormat(textFormat:TextFormat)`

  Applies the properties of `textFormat` to all text in the text field.

- **Usage 2:** `my_textField.setTextFormat(beginIndex:Number, textFormat:TextFormat)`

  Applies the properties of `textFormat` to the character at the `beginIndex` position.

- **Usage 3:** `my_textField.setTextFormat(beginIndex:Number, endIndex:Number, textFormat:TextFormat)`

  Applies the properties of the `textFormat` parameter to the span of text from the `beginIndex` position to the `endIndex` position.

Notice that any text inserted manually by the user receives the text field's default formatting for new text, and not the formatting specified for the text insertion point. To set a text field's default formatting for new text, use `TextField.setNewTextFormat()`.

Availability

Flash Lite 2.0

Parameters

beginIndex : Number [optional] - An integer that specifies the first character of the desired text span. If you do not specify `beginIndex` and `endIndex`, the TextFormat is applied to the entire TextField.

endIndex : Number [optional] - An integer that specifies the first character after the desired text span. If you specify `beginIndex` but do not specify `endIndex`, the TextFormat is applied to the single character specified by `beginIndex`.

textFormat : TextFormat - A TextFormat object, which contains character and paragraph formatting information.

Example

The following example sets the text format for two different strings of text. The `setTextFormat()` method is called and applied to the `my_txt` text field.

```
var format1_fmt:TextFormat = new TextFormat();
format1_fmt.font = "Arial";
var format2_fmt:TextFormat = new TextFormat();
format2_fmt.font = "Courier";

var string1:String = "Sample string number one."+newline;
var string2:String = "Sample string number two."+newline;

this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, 300, 200);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.text = string1;
var firstIndex:Number = my_txt.length;
my_txt.text += string2;
var secondIndex:Number = my_txt.length;

my_txt.setTextFormat(0, firstIndex, format1_fmt);
my_txt.setTextFormat(firstIndex, secondIndex, format2_fmt);
```

See also

getNewTextFormat (TextField.getNewTextFormat method), setNewTextFormat (TextField.setNewTextFormat method)

_soundbuftime (TextField._soundbuftime property)

```
public _soundbuftime : Number
```

Specifies the number of seconds a sound prebuffers before it starts to stream. Note: Although you can specify this property for a TextField object, it is actually a global property that applies to all sounds loaded, and you can specify its value simply as `_soundbuftime`. Setting this property for a TextField object actually sets the global property. For more information and an example, see `_soundbuftime`.

Availability

Flash Lite 2.0

See also

_soundbuftime property

tabEnabled (TextField.tabEnabled property)

```
public tabEnabled : Boolean
```

Specifies whether the text field is included in automatic tab ordering. It is `undefined` by default.

If the `tabEnabled` property is `undefined` or `true`, the object is included in automatic tab ordering. If the `tabIndex` property is also set to a value, the object is included in custom tab ordering as well. If `tabEnabled` is `false`, the object is not included in automatic or custom tab ordering, even if the `tabIndex` property is set.

Availability

Flash Lite 2.0

Example

The following example creates several text fields, called `one_txt`, `two_txt`, `three_txt` and `four_txt`. The `three_txt` text field has the `tabEnabled` property set to `false`, so it is excluded from the automatic tab ordering.

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100, 22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100, 22);
four_txt.border = true;
four_txt.type = "input";

three_txt.tabEnabled = false;
three_txt.text = "tabEnabled = false;";
```

See also

`tabEnabled (Button.tabEnabled property)`, `tabEnabled (MovieClip.tabEnabled property)`

tabIndex (TextField.tabIndex property)

```
public tabIndex : Number
```

Lets you customize the tab ordering of objects in a SWF file. You can set the `tabIndex` property on a button, movie clip, or text field instance; it is `undefined` by default.

If any currently displayed object in the SWF file contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the SWF file. The custom tab ordering only includes objects that have `tabIndex` properties.

The `tabIndex` property must be a positive integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` value of `1` precedes an object with a `tabIndex` value of 2. If two objects have the same `tabIndex` value, the one that precedes the other in the tab ordering is `undefined`.

The custom tab ordering defined by the `tabIndex` property is *flat*. This means that no attention is paid to the hierarchical relationships of objects in the SWF file. All objects in the SWF file with `tabIndex` properties are placed in the tab order, and the tab order is determined by the order of the `tabIndex` values. If two objects have the same `tabIndex` value, the one that goes first is `undefined`. You should not use the same `tabIndex` value for multiple objects.

Availability

Flash Lite 2.0

Example

The following ActionScript dynamically creates four text fields and assigns them to a custom tab order. Add the following ActionScript to your FLA or ActionScript file:

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100, 22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100, 22);
four_txt.border = true;
four_txt.type = "input";

one_txt.tabIndex = 3;
two_txt.tabIndex = 1;
three_txt.tabIndex = 2;
four_txt.tabIndex = 4;
```

See also

tabIndex (Button.tabIndex property), tabIndex (MovieClip.tabIndex property)

_target (TextField._target property)

```
public _target : String [read-only]
```

The target path of the text field instance. The `_self` target specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window.

Availability

Flash Lite 2.0

Example

The following ActionScript creates a text field called `my_txt` and outputs the target path of the new field, in both slash and dot notation.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
trace(my_txt._target); // output: /my_txt
trace(eval(my_txt._target)); // output: _level0.my_txt
```

text (TextField.text property)

```
public text : String
```

Indicates the current text in the text field. Lines are separated by the carriage return character ("\r", ASCII 13). This property contains the normal, unformatted text in the text field, without HTML tags, even if the text field is HTML.

Availability

Flash Lite 2.0

Example

The following example creates an HTML text field called `my_txt`, and assigns an HTML-formatted string of text to the field. When you trace the `htmlText` property, the Output panel displays the HTML-formatted string. When you trace the value of the `text` property, the unformatted string with HTML tags appears in the Output panel.When you trace the value of the `text` property, the unformatted string with HTML tags writes to the log file.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 400, 22);
my_txt.html = true;
my_txt.htmlText = "<b>Remember to always update the help panel.</b>";

trace("htmlText: "+my_txt.htmlText);
trace("text: "+my_txt.text);

// output:
htmlText: <P ALIGN="LEFT"><FONT FACE="Times New Roman" SIZE="12" COLOR="#000000">
<B>Remember to always update your help panel.</B></FONT></P>
text: Remember to always update your help panel.
```

See also

htmlText (TextField.htmlText property)

textColor (TextField.textColor property)

```
public textColor : Number
```

Indicates the color of the text in a text field. The hexadecimal color system uses six digits to represent color values. Each digit has sixteen possible values or characters. The characters range from 0 to 9 and then A to F. Black is represented by (`#000000`) and white, at the opposite end of the color system, is (`#FFFFFF`).

Availability

Flash Lite 2.0

Example

The following ActionScript creates a text field and changes its color property to red.

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);
my_txt.text = "this will be red text";
my_txt.textColor = 0xFF0000;
```

textHeight (TextField.textHeight property)

```
public textHeight : Number
```

Indicates the height of the text.

Availability

Flash Lite 2.0

Example

The following example creates a text field, and assigns a string of text to the field. A trace statement is used to display the text height and width in the Output panel. The `trace()` method is used to write the text height and width in the log file. The `autoSize` property is then used to resize the text field, and the new height and width will also be displayed in the Output panel. The `autoSize` property is then used to resize the text field, and the new height and width also write to the log file.

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);
my_txt.text = "Sample text";
trace("textHeight: "+my_txt.textHeight+", textWidth: "+my_txt.textWidth);
trace("_height: "+my_txt._height+", _width: "+my_txt._width+"\n");
my_txt.autoSize = true;
trace("after my_txt.autoSize = true;");
trace("_height: "+my_txt._height+", _width: "+my_txt._width);
```

Which outputs the following information:

```
textHeight: 15, textWidth: 56
_height: 300, _width: 100

after my_txt.autoSize = true;
_height: 19, _width: 60
```

See also

`textWidth (TextField.textWidth property)`

textWidth (TextField.textWidth property)

```
public textWidth : Number
```

Indicates the width of the text.

Availability

Flash Lite 2.0

Example

See the example for `TextField.textHeight`.

See also

`textHeight (TextField.textHeight property)`

type (TextField.type property)

```
public type : String
```

Specifies the type of text field. There are two values: `dynamic`, which specifies a dynamic text field that cannot be edited by the user, and `input`, which specifies an input text field.

Availability

Flash Lite 2.0

Example

The following example creates two text fields: `username_txt` and `password_txt`. Text is entered into both text fields; however, `password_txt` has the `password` property set to `true`. Therefore, the characters display as asterisks instead of as characters in the `password_txt` field.

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
username_txt.border = true;
username_txt.type = "input";
username_txt.maxChars = 16;
username_txt.text = "hello";

this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
password_txt.border = true;
password_txt.type = "input";
password_txt.maxChars = 16;
password_txt.password = true;
password_txt.text = "world";
```

_url (TextField._url property)

```
public _url : String [read-only]
```

Retrieves the URL of the SWF file that created the text field.

Availability

Flash Lite 2.0

Example

The following example retrieves the URL of the SWF file that created the text field, and the SWF file that loads into it.

```
this.createTextField("my_txt", 1, 10, 10, 100, 22);
trace(my_txt._url);

var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace(target_mc._url);
};
var holder_mcl:MovieClipLoader = new MovieClipLoader();
holder_mcl.addListener(mclListener);
holder_mcl.loadClip("best_flash_ever.swf", this.createEmptyMovieClip("holder_mc", 2));
```

When you test this example, the URL of the SWF file you are testing, and the file called best_flash_ever.swf are displayed in the Output panel.When you test this example, the URL of the SWF file you are testing, and the file called best_flash_ever.swf write to the log file.

variable (TextField.variable property)

```
public variable : String
```

The name of the variable that the text field is associated with. The type of this property is `String`.

Availability

Flash Lite 2.0

Example

The following example creates a text field called `my_txt` and associates the variable `today_date` with the text field. When you change the variable `today_date`, then the text that appears in `my_txt` updates.

```
this.(reateTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.variable = "today_date";
var today_date:Date = new Date();

var date_interval:Number = setInterval(updateDate, 500);
function updateDate():Void {
    today_date = new Date();
}
```

_visible (TextField._visible property)

```
public _visible : Boolean
```

A Boolean value that indicates whether the text field *my_txt* is visible. Text fields that are not visible (`_visible` property set to `false`) are disabled.

Availability

Flash Lite 2.0

Example

The following example creates a text field called `my_txt`. A button called `visible_btn` toggles the visibility(of `my_txt`.

```
this.createTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.background = true;
my_txt.backgroundColor = 0xDFDFDF;
my_txt.border = true;
my_txt.type = "input";

visible_btn.onRelease = function() {
    my_txt._visible = !my_txt._visible;
};
```

See also

`_visible (Button._visible property)`, `_visible (MovieClip._visible property)`

_width (TextField._width property)

```
public _width : Number
```

The width of the text field, in pixels.

Availability

Flash Lite 2.0

Example

The following example creates two text fields that you can use to change the width and height of a third text field on the Stage. Add the following ActionScript to a FLA or ActionScript file.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 40, 160, 120);
my_txt.background = true;
my_txt.backgroundColor = 0xFF0000;
my_txt.border = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.wordWrap = true;

this.createTextField("width_txt", this.getNextHighestDepth(), 10, 10, 30, 20);
width_txt.border = true;
width_txt.maxChars = 3;
width_txt.type = "input";
width_txt.text = my_txt._width;
width_txt.onChanged = function() {
    my_txt._width = this.text;
}

this.createTextField("height_txt", this.getNextHighestDepth(), 70, 10, 30, 20);
height_txt.border = true;
height_txt.maxChars = 3;
height_txt.type = "input";
height_txt.text = my_txt._height;
height_txt.onChanged = function() {
    my_txt._height = this.text;
}
```

When you test the example, try entering new values into `width_txt` and `height_txt` to change the dimensions of `my_txt`.

See also

`_height (TextField._height property)`

wordWrap (TextField.wordWrap property)

```
public wordWrap : Boolean
```

A Boolean value that indicates if the text field has word wrap. If the value of `wordWrap` is `true`, the text field has word wrap; if the value is `false`, the text field does not have word wrap.

Availability

Flash Lite 2.0

Example

The following example demonstrates how `wordWrap` affects long text in a text field that is created at runtime.

```
this.createTextField("my_txt", 99, 10, 10, 100, 200);
my_txt.text = "This is very long text that will certainly extend beyond the width of this text
field";
my_txt.border = true;
```

Test the SWF file in Flash Lite player by selecting Control > Test Movie. Then return to your ActionScript, and add the following line to the code and test the SWF file again:

```
my_txt.wordWrap = true;
```

_x (TextField._x property)

```
public _x : Number
```

An integer that sets the *x* coordinate of a text field relative to the local coordinates of the parent movie clip. If a text field is on the main Timeline, then its coordinate system refers to the upper-left corner of the Stage as (0, 0). If the text field is inside a movie clip that has transformations, the text field is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90 degrees counterclockwise, the enclosed text field inherits a coordinate system that is rotated 90 degrees counterclockwise. The text field's coordinates refer to the registration point position.

Availability

Flash Lite 2.0

Example

The following example creates a text field wherever you click the mouse. When it creates a text field, that field displays the current *x* and *y* coordinates of the text field.

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 60, 22);
coords_txt.autoSize = true;
coords_txt.selectable = false;
coords_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    coords_txt.text = "X:"+Math.round(_xmouse)+", Y:"+Math.round(_ymouse);
    coords_txt._x = _xmouse;
    coords_txt._y = _ymouse;
};
Mouse.addListener(mouseListe(er);
```

See also

_xscale (TextField._xscale property), _y (TextField._y property), _yscale (TextField._yscale property)

_xmouse (TextField._xmouse property)

```
public _xmouse : Number [read-only]
```

Returns the *x* coordinate of the mouse position relative to the text field.

Note: This property is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability

Flash Lite 2.0

Example

The following example creates three text fields on the Stage. The `mouse_txt` instance displays the current position of the mouse in relation to the Stage. The `textfield_txt` instance displays the current position of the mouse pointer in relation to the `my_txt` instance. Add the following ActionScript to a FLA or ActionScript file:

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 10, 10, 200, 22);
mouse_txt.border = true;
this.createTextField("textfield_txt", this.getNextHighestDepth(), 220, 10, 200, 22);
textfield_txt.border = true;
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100, 160, 120);
my_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    mouse_txt.text = "MOUSE ... X:" + Math.round(_xmouse) + ",\tY:" + Math.round(_ymouse);
    textfield_txt.text = "TEXTFIELD ... X:" + Math.round(my_txt._xmouse) + ",\tY:" +
    Math.round(my_txt._ymouse);
}

Mouse.addListener(mouseListener);
```

See also

_ymouse (TextField._ymouse property)

_xscale (TextField._xscale property)

```
public _xscale : Number
```

Determines the horizontal scale of the text field as applied from the registration point of the text field, expressed as a percentage. The default registration point is (0,0).

Availability

Flash Lite 2.0

Example

The following example scales the `my_txt` instance when you click the `scaleUp_btn` and `scaleDown_btn` instances.

```
this.createTextField("my_txt", 99, 10, 40, 100, 22);
my_txt.autoSize = true;
my_txt.border = true;
my_txt.selectable = false;
my_txt.text = "Sample text goes here.";

scaleUp_btn.onRelease = function() {
    my_txt._xscale = 2;
    my_txt._yscale = 2;
}
scaleDown_btn.onRelease = function() {
    my_txt._xscale /= 2;
    my_txt._yscale /= 2;
}
```

See also

_x (TextField._x property), _y (TextField._y property), _yscale (TextField._yscale property)

_y (TextField._y property)

```
public _y : Number
```

The *y* coordinate of a text field relative to the local coordinates of the parent movie clip. If a text field is in the main Timeline, then its coordinate system refers to the upper-left corner of the Stage as (0, 0). If the text field is inside another movie clip that has transformations, the text field is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90 degrees counterclockwise, the enclosed text field inherits a coordinate system that is rotated 90 degrees counterclockwise. The text field's coordinates refer to the registration point position.

Availability
Flash Lite 2.0

Example
See the example for `TextField._x`.

See also
`_x (TextField._x property)`, `_xscale (TextField._xscale property)`, `_yscale (TextField._yscale property)`

_ymouse (TextField._ymouse property)

```
public _ymouse : Number [read-only]
```

Indicates the *y* coordinate of the mouse position relative to the text field.

Note: This property is supported in Flash Lite only if `System.capabilities.hasMouse` is `true` or `System.capabilities.hasStylus` is `true`.

Availability
Flash Lite 2.0

Example
See the example for `TextField._xmouse`.

See also
`_xmouse (TextField._xmouse property)`

_yscale (TextField._yscale property)

```
public _yscale : Number
```

The vertical scale of the text field as applied from the registration point of the text field, expressed as a percentage. The default registration point is (0,0).

Availability
Flash Lite 2.0

Example
See the example for `TextField._xscale`.

See also
`_x (TextField._x property)`, `_xscale (TextField._xscale property)`, `_y (TextField._y property)`

TextFormat

```
Object
    |
    +-TextFormat
```

```
public class TextFormat
extends Object
```

The TextFormat class represents character formatting information. Use the TextFormat class to create specific text formatting for text fields. You can apply text formatting to both static and dynamic text fields. Some properties of the TextFormat class are not available for both embedded and device fonts.

Availability

Flash Lite 2.0

See also

setTextFormat (TextField.setTextFormat method), getTextFormat (TextField.getTextFormat method)

Property summary

Modifiers	Property	Description
	align : String	A string that indicates the alignment of the paragraph.
	blockIndent : Number	A number that indicates the block indentation in points.
	bold : Boolean	A Boolean value that specifies whether the text is boldface.
	bullet : Boolean	A Boolean value that indicates that the text is part of a bulleted list.
	color : Number	A number that indicates the color of text.
	font : String	A string that specifies the name of the font for text.
	indent : Number	An integer that indicates the indentation from the left margin to the first character in the paragraph.
	italic : Boolean	A Boolean value that indicates whether text in this text format is italicized.
	leading : Number	An integer that represents the amount of vertical space in pixels (called *leading*) between lines.
	leftMargin : Number	The left margin of the paragraph, in points.
	rightMargin : Number	The right margin of the paragraph, in points.
	size : Number	The point size of text in this text format.
	tabStops : Array	Specifies custom tab stops as an array of non-negative integers.
	target : String	Indicates the target window where the hyperlink is displayed.
	underline : Boolean	A Boolean value that indicates whether the text that uses this text format is underlined (`true`) or not (`false`).
	url : String	Indicates the URL to which text in this text format hyperlinks.

Properties inherited from class Object

constructor (Object.constructor property), __proto__ (Object.__proto__ property), prototype (Object.prototype property), __resolve (Object.__resolve property)

Constructor summary

Signature	Description
TextFormat([font:String], [size:Number], [color:Number], [bold:Boolean], [italic:Boolean], [underline:Boolean], [url:String], [target:String], [align:String], [leftMargin:Number], [rightMargin:Number], [indent:Number], [leading:Number])	Creates a TextFormat object with the specified properties.

Method summary

Modifiers	Signature	Description
	getTextExtent(text:String, [width:Number]) : Object	Returns text measurement information for the text string `text` in the format specified by `my_fmt`.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

align (TextFormat.align property)

```
public align : String
```

A string that indicates the alignment of the paragraph. You can apply this property to static and dynamic text. The following list shows possible values for this property:

- `"left"`—the paragraph is left-aligned.
- `"center"`—the paragraph is centered.
- `"right"`—the paragraph is right-aligned.

The default value is `null`, which indicates that the property is undefined.

Availability

Flash Lite 2.0

Example

The following example creates a text field with a border and uses `TextFormat.align` to center the text.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.align = "center";

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first text field object text";
my_txt.setTextFormat(my_fmt);
```

blockIndent (TextFormat.blockIndent property)

```
public blockIndent : Number
```

A number that indicates the block indentation in points. Block indentation is applied to an entire block of text; that is, to all lines of the text. In contrast, normal indentation (`TextFormat.indent`) affects only the first line of each paragraph. If this property is `null`, the TextFormat object does not specify block indentation.

Availability

Flash Lite 2.0

Example

This example creates a text field with a border and sets the `blockIndent` to 20.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.blockIndent = 20;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

bold (TextFormat.bold property)

```
public bold : Boolean
```

A Boolean value that specifies whether the text is boldface. The default value is `null`, which indicates that the property is undefined. If the value is `true`, the text is boldface.

Note: For Arabic, Hebrew, and Thai, this property works for paragraph-level formatting only.

Availability

Flash Lite 2.0

Example

The following example creates a text field that includes characters in boldface.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "This is my text field object text";
my_txt.setTextFormat(my_fmt);
```

bullet (TextFormat.bullet property)

```
public bullet : Boolean
```

A Boolean value that indicates that the text is part of a bulleted list. In a bulleted list, each paragraph of text is indented. To the left of the first line of each paragraph, a bullet symbol is displayed. The default value is `null`.

Note: For Flash Lite, this property works for embedded fonts only. This property is not supported for Arabic, Hebrew, and Thai.

Availability

Flash Lite 2.0

Example

The following example creates a new text field at runtime, and puts a string with a line break into the field. The TextFormat class is used to format the characters by adding bullets to each line in the text field. This is demonstrated in the following ActionScript:

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bullet = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my text"+newline;
my_txt.text += "this is more text"+newline;
my_txt.setTextFormat(my_fmt);
```

color (TextFormat.color property)

```
public color : Number
```

A number that indicates the color of text. The number contains three 8-bit RGB components; for example, 0xFF0000 is red, and 0x00FF00 is green.

Note: For Arabic, Hebrew, and Thai, this property works for paragraph-level formatting only.

Availability

Flash Lite 2.0

Example

The following example creates a text field and sets the text color to red.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.blockIndent = 20;
my_fmt.color = 0xFF0000; // hex value for red

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first text field object text";
my_txt.setTextFormat(my_fmt);
```

font (TextFormat.font property)

```
public font : String
```

A string that specifies the name of the font for text. The default value is `null`, which indicates that the property is undefined.

Note: For Flash Lite, this property works for embedded fonts only. This property is not supported for Arabic, Hebrew, and Thai.

Availability

Flash Lite 2.0

Example

The following example creates a text field and sets the font to Courier.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.font = "Courier";

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);
```

getTextExtent (TextFormat.getTextExtent method)

```
public getTextExtent(text:String, [width:Number]) : Object
```

Returns text measurement information for the text string `text` in the format specified by `my_fmt`. The text string is treated as plain text (not HTML).

The method returns an object with six properties: `ascent`, `descent`, `width`, `height`, `textFieldHeight`, and `textFieldWidth`. All measurements are in pixels.

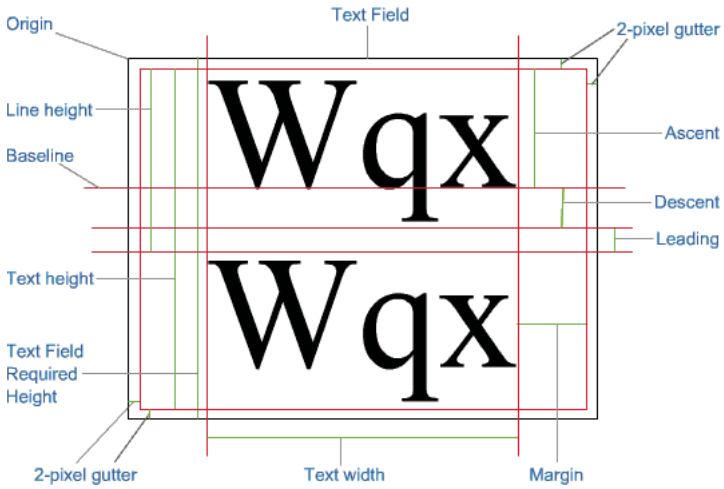
If a `width` parameter is specified, word wrapping is applied to the specified text. This lets you determine the height at which a text box shows all of the specified text.

The `ascent` and `descent` measurements provide, respectively, the distance above and below the baseline for a line of text. The baseline for the first line of text is positioned at the text field's origin plus its `ascent` measurement.

The `width` and `height` measurements provide the width and height of the text string. The `textFieldHeight` and `textFieldWidth` measurements provide the height and width required for a text field object to display the entire text string. Text fields have a 2-pixel-wide gutter around them, so the value of `textFieldHeight` is equal the value of `height` + 4; likewise, the value of `textFieldWidth` is always equal to the value of `width` + 4.

If you are creating a text field based on the text metrics, use `textFieldHeight` rather than `height` and `textFieldWidth` rather than `width`.

The following figure illustrates these measurements:



When setting up your TextFormat object, set all the attributes exactly as they will be set for the creation of the text field, including font name, font size, and leading. The default value for leading is 2.

Availability

Flash Lite 2.0

Parameters

text : String - A string.

width : Number [optional] - A number that represents the width, in pixels, at which the specified text should wrap.

Returns

Object - An object with the properties `width`, `height`, `ascent`, `descent`, `textFieldHeight`, `textFieldWidth`.

Example

This example creates a single-line text field that's just big enough to display a text string using the specified formatting.

```
var my_str:String = "Small string";

// Create a TextFormat object,
// and apply its properties.
var my_fmt:TextFormat = new TextFormat();
with (my_fmt) {
    font = "Arial";
    bold = true;
}

// Obtain metrics information for the text string
// with the specified formatting.
var metrics:Object = my_fmt.getTextExtent(my_str);

// Create a text field just large enough to display the text.
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100, metrics.textFieldWidth,
metrics.textFieldHeight);
my_txt.border = true;
my_txt.wordWrap = true;
// Assign the same text string and TextFormat object to the my_txt object.
my_txt.text = my_str;
my_txt.setTextFormat(my_fmt);
```

The following example creates a multiline, 100-pixel-wide text field that's high enough to display a string with the specified formatting.

```
// Create a TextFormat object.
var my_fmt:TextFormat = new TextFormat();
// Specify formatting properties for the TextFormat object:
my_fmt.font = "Arial";
my_fmt.bold = true;
my_fmt.leading = 4;

// The string of text to be displayed
var textToDisplay:String = "Adobe Flash Player 7, now with improved text metrics.";

// Obtain text measurement information for the string,
// wrapped at 100 pixels.
var metrics:Object = my_fmt.getTextExtent(textToDisplay, 100);

// Create a new TextField object using the metric
// information just obtained.
this.createTextField("my_txt", this.getNextHighestDepth(), 50, 50-metrics.ascent, 100,
metrics.textFieldHeight);
my_txt.wordWrap = true;
my_txt.border = true;
// Assign the text and the TextFormat object to the TextObject:
my_txt.text = textToDisplay;
my_txt.setTextFormat(my_fmt);
```

indent (TextFormat.indent property)

```
public indent : Number
```

An integer that indicates the indentation from the left margin to the first character in the paragraph. The default value is `null`, which indicates that the property is undefined.

Availability
Flash Lite 2.0

Example
The following example creates a text field and sets the indentation to 10:

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.indent = 10;

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);
```

See also
blockIndent (TextFormat.blockIndent property)

italic (TextFormat.italic property)

```
public italic : Boolean
```

A Boolean value that indicates whether text in this text format is italicized. The default value is `null`, which indicates that the property is undefined.

Note: For Arabic, Hebrew, and Thai, this property works for paragraph-level formatting only.

Availability
Flash Lite 2.0

Example
The following example creates a text field and sets the text style to italic.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.italic = true;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

leading (TextFormat.leading property)

```
public leading : Number
```

An integer that represents the amount of vertical space in pixels (called *leading*) between lines. The default value is `null`, which indicates that the property is undefined.

Availability
Flash Lite 2.0

Example
The following example creates a text field and sets the leading to 10.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.leading = 10;

this.createTextField("my_txt", 1, 100, 100, 100, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "This is my first text field object text";
my_txt.setTextFormat(my_fmt);
```

leftMargin (TextFormat.leftMargin property)

```
public leftMargin : Number
```

The left margin of the paragraph, in points. The default value is `null`, which indicates that the property is undefined.

Availability
Flash Lite 2.0

Example
The following example creates a text field and sets the left margin to 20 points.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.leftMargin = 20;

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);
```

rightMargin (TextFormat.rightMargin property)

```
public rightMargin : Number
```

The right margin of the paragraph, in points. The default value is `null`, which indicates that the property is undefined.

Availability
Flash Lite 2.0

Example
The following example creates a text field and sets the right margin to 20 points.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.rightMargin = 20;

mytext.text = "this is my first text field object text";
mytext.setTextFormat(myformat);
```

size (TextFormat.size property)

```
public size : Number
```

The point size of text in this text format. The default value is `null`, which indicates that the property is undefined.

Note: For Arabic, Hebrew, and Thai, this property works for paragraph-level formatting only.

Availability

Flash Lite 2.0

Example

The following example creates a text field and sets the text size to 20 points.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.size = 20;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

tabStops (TextFormat.tabStops property)

```
public tabStops : Array
```

Specifies custom tab stops as an array of non-negative integers. Each tab stop is specified in pixels. If custom tab stops are not specified (`null`), the default tab stop is 4 (average character width).

Note: For Flash Lite, this property works for embedded fonts only. This property is not supported for Arabic, Hebrew, and Thai.

Availability

Flash Lite 2.0

Example

The following example creates two text fields, one with tab stops every 40 pixels, and the other with tab stops every 75 pixels.

```
this.createTextField("mytext",1,100,100,400,100);
mytext.border = true;
var myformat:TextFormat = new TextFormat();
myformat.tabStops = [40,80,120,160];
mytext.text = "A\tB\tC\tD"; // \t is the tab stop character
mytext.setTextFormat(myformat);

this.createTextField("mytext2",2,100,220,400,100);
mytext2.border = true;
var myformat2:TextFormat = new TextFormat();
myformat2.tabStops = [75,150,225,300];
mytext2.text ="A\tB\tC\tD";
mytext2.setTextFormat(myformat2);
```

target (TextFormat.target property)

```
public target : String
```

Indicates the target window in which the hyperlink is displayed. If the target window is an empty string, the text is displayed in the default target window `_self`. You can choose a custom name or one of the following four names: `_self` specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window. If the `TextFormat.url` property is an empty string or `null`, you can get or set this property, but the property will have no effect.

Availability

Flash Lite 2.0

Example

The following example creates a text field with a hyperlink to the Adobe website. The example uses `TextFormat.target` to display the Adobe website in a new browser window.

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.adobe.com";
myformat.target = "_blank";

this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Adobe.com";
mytext.setTextFormat(myformat);
```

See also

url (TextFormat.url property)

TextFormat constructor

```
public TextFormat([font:String], [size:Number], [color:Number], [bold:Boolean],
[italic:Boolean], [underline:Boolean], [url:String], [target:String], [align:String],
[leftMargin:Number], [rightMargin:Number], [indent:Number], [leading:Number])
```

Creates a TextFormat object with the specified properties. You can then change the properties of the TextFormat object to change the formatting of text fields.

Any parameter may be set to `null` to indicate that it is not defined. All of the parameters are optional; any omitted parameters are treated as `null`.

Availability
Flash Lite 2.0

Parameters
font : String [optional] - The name of a font for text as a string.

size : Number [optional] - An integer that indicates the point size.

color : Number [optional] - The color of text using this text format. A number containing three 8-bit RGB components; for example, 0xFF0000 is red, and 0x00FF00 is green.

bold : Boolean [optional] - A Boolean value that indicates whether the text is boldface.

italic : Boolean [optional] - A Boolean value that indicates whether the text is italicized.

underline : Boolean [optional] - A Boolean value that indicates whether the text is underlined.

url : String [optional] - The URL to which the text in this text format hyperlinks. If `url` is an empty string, the text does not have a hyperlink.

target : String [optional] - The target window in which the hyperlink is displayed. If the target window is an empty string, the text is displayed in the default target window `_self`. If the `url` parameter is set to an empty string or to the value `null`, you can get or set this property, but the property will have no effect.

align : String [optional] - The alignment of the paragraph, represented as a string. If `"left"`, the paragraph is left-aligned. If `"center"`, the paragraph is centered. If `"right"`, the paragraph is right-aligned.

leftMargin : Number [optional] - Indicates the left margin of the paragraph, in points.

rightMargin : Number [optional] - Indicates the right margin of the paragraph, in points.

indent : Number [optional] - An integer that indicates the indentation from the left margin to the first character in the paragraph.

leading : Number [optional] - A number that indicates the amount of leading vertical space between lines.

Availability
Flash Lite 2.0

Example
The following example creates a TextFormat object, formats the `stats_txt` text field, and creates a new text field in which to display the text:

```
// Define a TextFormat which is used to format the stats_txt text field.
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;
my_fmt.font = "Arial";
my_fmt.size = 12;
my_fmt.color = 0xFF0000;
// Create a text field to display the player's statistics.
this.createTextField("stats_txt", 5000, 10, 0, 530, 22);
// Apply the TextFormat to the text field.
stats_txt.setNewTextFormat(my_fmt);
stats_txt.selectable = false;
stats_txt.text = "Lorem ipsum dolor sit amet...";
```

To view another example, see the animations.fla file in the ActionScript samples folder at The Adobe Flash samples page. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

underline (TextFormat.underline property)

```
public underline : Boolean
```

A Boolean value that indicates whether the text that uses this text format is underlined (`true`) or not (`false`). This underlining is similar to that produced by the `<U>` tag, but the latter is not true underlining, because it does not skip descenders correctly. The default value is `null`, which indicates that the property is undefined.

Note: For Arabic, Hebrew, and Thai, this property works for paragraph-level formatting only.

Availability

Flash Lite 2.0

Example

The following example creates a text field and sets the text style to underline.

```
this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.underline = true;
mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

url (TextFormat.url property)

```
public url : String
```

Indicates the URL to which text in this text format hyperlinks. If the `url` property is an empty string, the text does not have a hyperlink. The default value is `null`, which indicates that the property is undefined.

Availability

Flash Lite 2.0

Example

This example creates a text field that is a hyperlink to the Adobe website.

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.adobe.com";

this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Adobe.com";
mytext.setTextFormat(myformat);
```

Transform (flash.geom.Transform)

```
Object
    |
    +-flash.geom.transform

public class Transform
extends Object
```

The Transform class collects data about color transformations and coordinate manipulations that are applied to a MovieClip object.

A Transform object is normally obtained by getting the value of the `transform` property from a MovieClip object.

Availability

Flash Lite 3.1

See also

`transform (MovieClip.transform property)`, `ColorTransform (flash.geom.ColorTransform)`, `Matrix (flash.geom.Matrix)`

Property summary

Modifiers	Property	Description
	colorTransform : ColorTransform	A ColorTransform object containing values that universally adjust the colors in the movie clip.
	concatenatedColorTransform : ColorTransform `[read-only]`	A ColorTransform object representing the combined color transformations applied to this object and all of its parent objects, back to the root level.
	ConcatenatedMatrix : Matrix `[read-only]`	A Matrix object representing the combined transformation matrixes of this object and all of its parent objects, back to the root level.
	matrix : Matrix	A transformation Matrix object containing values that affect the scaling, rotation, and translation of the movie clip.
	pixelBounds : Rectangle	A Rectangle object that defines the bounding rectangle of the MovieClip object on the Stage.

constructor (Object.constructor property), __proto__ (Object.__proto__ property), "prototype (Object.prototype property)" on page 499, __resolve (Object.__resolve property)

Constructor summary

Signature	Description
Transform (mc : MovieClip)	Creates a new Transform object attached to the given MovieClip object.

Method summary

"addProperty (Object.addProperty method)" on page 494, "hasOwnProperty (Object.hasOwnProperty method)" on page 497, "isPropertyEnumerable (Object.isPropertyEnumerable method)" on page 497, "isPrototypeOf (Object.isPrototypeOf method)" on page 498, "registerClass (Object.registerClass method)" on page 500, "toString (Object.toString method)" on page 504, "unwatch (Object.unwatch method)" on page 505, "valueOf (Object.valueOf method)" on page 506, "watch (Object.watch method)" on page 507

colorTransform (Transform.colorTransform property)

```
public colorTransform : ColorTransform
```

A ColorTransform object containing values that universally adjust the colors in the movie clip.

Availability

Flash Lite 3.1

Example

The following example applies the ColorTransform object `blueColorTransform` to the Transform object `trans`. This ColorTransform converts the color of the MovieClip `rect` from red to blue.

```
import flash.geom.Transform;
import flash.geom.ColorTransform;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var trans:Transform = new Transform(rect);
trace(trans.colorTransform);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=0,
greenOffset=0, blueOffset=0, alphaOffset=0)

var blueColorTransform:ColorTransform = new ColorTransform(0, 1, 1, 1, 0, 0, 255, 0);

rect.onPress = function() {
    trans.colorTransform = blueColorTransform;
    trace(trans.colorTransform);
    // (redMultiplier=0, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=0,
greenOffset=0, blueOffset=255, alphaOffset=0)
}

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
        mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

See also

"ColorTransform (flash.geom.ColorTransform)" on page 263

concatenatedColorTransform (Transform.concatenatedColorTransform property)

`public concatenatedColorTransform : ColorTransform [read-only]`

A ColorTransform object representing the combined color transformations applied to this object and all of its parent objects, back to the root level. If different color transformations have been applied at different levels, each of those transformations will be concatenated into one ColorTransform object for this property.

Availability

Flash Lite 3.1

Example

The following example applies two Transform objects to both a parent and child MovieClip object. A `blueColorTransform` variable is then applied to the Transform object `parentTrans`, which adjusts the color of both parent and child MovieClip objects toward blue. You can see how `child.concatenatedColorTransform` is the combination of `parentTrans` and `childTrans`.

```
import flash.geom.Transform;
import flash.geom.ColorTransform;

var parentRect:MovieClip = createRectangle(20, 80, 0xFF0000);
var childRect:MovieClip = createRectangle(10, 40, 0x00FF00, parentRect);

var parentTrans:Transform = new Transform(parentRect);
var childTrans:Transform = new Transform(childRect);

var blueColorTransform:ColorTransform = new ColorTransform(0, 1, 1, 1, 0, 0, 255, 0);

parentTrans.colorTransform = blueColorTransform;

trace(childTrans.concatenatedColorTransform);
// (redMultiplier=0, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=0,
greenOffset=0, blueOffset=255, alphaOffset=0)
trace(childTrans.colorTransform);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=0,
greenOffset=0, blueOffset=0, alphaOffset=0)
trace(parentTrans.concatenatedColorTransform);
// (redMultiplier=0, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1, redOffset=0,
greenOffset=0, blueOffset=255, alphaOffset=0)

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

See also

"ColorTransform (flash.geom.ColorTransform)" on page 263

concatenatedMatrix (Transform.concatenatedMatrix property)

```
public concatenatedMatrix : Matrix [read-only]
```

A Matrix object representing the combined transformation matrixes of this object and all of its parent objects, back to the root level. If different transformation matrixes have been applied at different levels, each of those matrixes will be concatenated into one matrix for this property.

Availability

Flash Lite 3.1

Example

The following example applies two Transform objects to both a child and parent MovieClip object. A `scaleMatrix` is then applied to the Transform object `parentTrans`, which scales both parent and child MovieClip objects. You can see how `child.concatenatedMatrix` is the combination of `parentTrans` and `childTrans`.

```
import flash.geom.Transform;
import flash.geom.Matrix;

var parentRect:MovieClip = createRectangle(20, 80, 0xFF0000);
var childRect:MovieClip = createRectangle(10, 40, 0x00FF00, parentRect);

var parentTrans:Transform = new Transform(parentRect);
var childTrans:Transform = new Transform(childRect);

var scaleMatrix:Matrix = new Matrix();
scaleMatrix.scale(2, 2);

parentTrans.matrix = scaleMatrix;

trace(childTrans.concatenatedMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(childTrans.matrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
trace(parentTrans.concatenatedMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

matrix (Transform.matrix property)

```
public matrix : Matrix
```

A transformation Matrix object containing values that affect the scaling, rotation, and translation of the movie clip.

Availability

Flash Lite 3.1

Example

The following example applies the Matrix object `scaleMatrix` to the Transform object `trans`. This Matrix scales the MovieClip `rect` by a factor of two.

```
import flash.geom.Transform;
import flash.geom.Matrix;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var trans:Transform = new Transform(rect);
trace(trans.matrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var scaleMatrix:Matrix = new Matrix();
scaleMatrix.scale(2, 2);

rect.onPress() = function() {
    trans.matrix = scaleMatrix;
    trace(trans.matrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
}

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

See also

"Matrix (flash.geom.Matrix)" on page 362

pixelBounds (Transform.pixelBounds property)

```
public pixelBounds : Rectangle
```

A Rectangle object that defines the bounding rectangle of the MovieClip object on the Stage.

Availability

Flash Lite 3.1

Example

The following example creates a Transform object `trans` and traces out its `pixelBounds` property. Notice that `pixelBounds` returns a bounding box with values equal to the MovieClip object's `getBounds()` and `getRect()` methods.

```
import flash.geom.Transform;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);
var trans:Transform = new Transform(rect);
trace(trans.pixelBounds); // (x=0, y=0, w=20, h=80)

var boundsObj:Object = rect.getBounds();
trace(boundsObj.xMin); // 0
trace(boundsObj.yMin); // 0
trace(boundsObj.xMax); // 20
trace(boundsObj.yMax); // 80

var rectObj:Object = rect.getRect();
trace(rectObj.xMin); // 0
trace(rectObj.yMin); // 0
trace(rectObj.xMax); // 20
trace(rectObj.yMax); // 80

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

Transform constructor

```
public Transform(mc:MovieClip)
```

Creates a new Transform object attached to the given MovieClip object.

When it is created, the new Transform object can be retrieved by getting the `transform` property of the given MovieClip object.

Availability

Flash Lite 3.1

Parameters

mc: MovieClip - The MovieClip object to which the new Transform object is applied.

Example

The following example creates the Transform `trans` and applies it to the MovieClip `rect`. You can see that the Transform object's `trans` and `rect.transform` do not evaluate as equals even though they contain the same values.

```
import flash.geom.Transform;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var trans:Transform = new Transform(rect);

trace(rect.transform == trans); // false

for(var i in trans) {
    trace(">> " + i + ": " + trans[i]);
    // >> pixelBounds: (x=0, y=0, w=20, h=80)
    // >> concatenatedColorTransform: (redMultiplier=1, greenMultiplier=1, blueMultiplier=1,
alphaMultiplier=1, redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)
    // >> colorTransform: (redMultiplier=1, greenMultiplier=1, blueMultiplier=1,
alphaMultiplier=1, redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)
    // >> concatenatedMatrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
    // >> matrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
}

for(var i in rect.transform) {
    trace(">> " + i + ": " + rect.transform[i]);
    // >> pixelBounds: (x=0, y=0, w=20, h=80)
    // >> concatenatedColorTransform: (redMultiplier=1, greenMultiplier=1, blueMultiplier=1,
alphaMultiplier=1, redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)
    // >> colorTransform: (redMultiplier=1, greenMultiplier=1, blueMultiplier=1,
alphaMultiplier=1, redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)
    // >> concatenatedMatrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
    // >> matrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
}

function createRectangle(width:Number, height:Number, color:Number, scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

Video

```
Object
    |
    +-Video

public class Video
extends Object
```

The Video class enables you to display video content that is embedded in your SWF file, stored locally on the host device, or streamed in from a remote location.

Note: The player for Flash Lite 2.0 handles video differently than Flash Player 7 does. These are the major differences:

- Flash Player 7 directly renders the video data (embedded or streaming). The player for Flash Lite 2.0 does not render the video data; instead it hands off the data to the mobile device. The player for Flash Lite 3.0 supports the rendering of Flash Video (FLV) directly by Flash Lite.
- Flash Player 7 supports many video formats in addition to FLV. Flash Lite 2.0 supports video playback in the following cases: video embedded in a SWF file; video that resides in a separate file on the host device; and video data that is streamed in over the network (in real time). The player for Flash Lite 2.0 supports only those video formats that a specific mobile device supports, while the player for Flash Lite 3.0 supports the rendering of FLV natively.
- Flash Player 7 lets you bundle the data in a SWF file or stream it by using the Video object and assigning either a NetStream object or Camera object as the source of the video information. However, the player for Flash Lite 2.0 does not support the `NetStream` and Camera objects. Instead, Flash Lite 2.0 uses a new library symbol type called `Video` to embed source video data and to stream video for mobile devices. Because Flash Lite 2.0 does not support the NetStream object, you use the methods and properties of the Video class to control the video playback. The player for Flash Lite 3.0 does support the `NetStream` and NetConnection objects, and you use the methods and properties of these classes to control FLV playback. Flash Lite 3.0 also supports a new property in the Video class, `attachVideo`, that specifies a video stream to be displayed within the Video object on the Stage. Flash Lite 3.0 does not support the Camera object.

Because of the requirements of mobile devices (smaller processor speeds, memory restrictions, and proprietary encoding formats), Flash Lite 2.0 cannot render the video information directly. The supported file formats for video depend on the mobile device manufacturer. For more information about supported video formats, check the hardware platforms on which you plan to deploy your application. In contrast, Flash Lite 3.0 does render Flash video directly.

Flash Lite 2.0 does not support the following Flash Player 7 features:

- Streaming of video data from a Flash Media Server
- Recording video

Flash Lite 3.0 adds support the following Flash Player 7 features:

- Flash video rendered directly by the player using versions of the On2 and Sorenson codecs optimized for mobile devices
- Streaming of video data over an RTMP (Real Time Messaging Protocol) connection to a Flash Media Server. (RTMPT (Real Time Messaging Protocol Tunnel) and RTMPS (Real Time Messaging Protocol Secure) connections are not supported, nor are multiple connections.)

Flash Lite 3.0 does not support the following Flash Player 7 features:

- Recording video
- Camera object

Availability

Flash Lite 2.0

Property summary

Properties inherited from class Object

```
constructor (Object.constructor property),__proto__ (Object.__proto__ property),
prototype (Object.prototype property),__resolve (Object.__resolve property)
```

Event summary

Event	Description
onStatus = function(infoObject :Object) {}	Callback handler that can be invoked by the device to indicate status or error conditions.

Method summary

Modifiers	Signature	Description
	close() : Void	Stops playback of the video, frees the memory associated with this Video object, and clears the video area onscreen.
	pause() : Void	Stops playback of the video and continues to render the current frame onscreen.
	play() : Boolean	Calling this method opens a video source and begins playback of a video.
	resume() : Void	Calling this method resumes playback of the video.
	stop() : Void	Stops playback of the video and continues to render the current frame onscreen.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method), isPropertyEnumerable (Object.isPropertyEnumerable method), isPrototypeOf (Object.isPrototypeOf method), registerClass (Object.registerClass method), toString (Object.toString method), unwatch (Object.unwatch method), valueOf (Object.valueOf method), watch (Object.watch method)

attachVideo (Video.attachVideo method)

```
public attachVideo(source:Object) : Void
```

Specifies a video stream (source) to be displayed within the boundaries of the Video object on the Stage. The video stream is either an FLV file being displayed by means of the `NetStream.play()` command, or `null`. If `source` is `null`, video is no longer played within the Video object.

You don't have to use this method if the FLV file contains only audio; the audio portion of FLV files is played automatically when the `NetStream.play()` command is issued.

If you want to control the audio associated with an FLV file, you can use `MovieClip.attachAudio()` to route the audio to a movie clip; you can then create a Sound object to control some aspects of the audio. For more information, see `MovieClip.attachAudio()`.

Availability

Flash Lite 3.0

Example

The following example plays a previously recorded file named video1.flv that is stored in the same directory as the SWF file.

```
var my_video:Video; // my_video is a Video object on the Stage
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
```

See also

play (Video.play method), stop (Video.stop method), resume (Video.resume method)

close (Video.close method)

```
public close() : Void
```

Stops playback of the video, frees the memory associated with this Video object, and clears the video area onscreen.

Availability

Flash Lite 2.0

Example

The following example closes the video that is playing in a Video object named `video1`.

```
video1.close()
```

See also

play (Video.play method), pause (Video.pause method), resume (Video.resume method)

onStatus (Video.onStatus handler)

```
onStatus = function(infoObject:Object) {}
```

Callback handler that can be invoked by the device to indicate status or error conditions.

Availability

Flash Lite 3.0

Parameters

infoObject : Object - The `infoObject` parameter has two properties:

- `code:String` — Description of the error or status condition (device specific).
- `level:Number` — Zero for error and non-zero for success (device specific).

Example

The following example shows how to create a `Video.onStatus()` function that displays a status or error condition.

```
var v:Video; // v is a Video object on the stage.
v.onStatus = function(o:Object)
{
    if ( o.level )
    {
        trace( "Video Status Msg (" + o.level + "): " + o.code );
    }
    else
    {
        trace( "Video Status Error: " + o.code );
    }
}
v.play("a.vid");
```

pause (Video.pause method)

```
public pause() : Void
```

Stops playback of the video and continues to render the current frame onscreen. A subsequent call to `Video.resume()` resumes playback from the current position.

Availability

Flash Lite 2.0

Example

The following example stops the video that is playing in a Video object (called `my_video`) when the user clicks the `close_btn` instance.

```
// video1 is the name of a Video object on Stage
video1.pause()
```

See also

`play (Video.play method)`, `stop (Video.stop method)`, `resume (Video.resume method)`

play (Video.play method)

```
public play() : Boolean
```

Calling this method opens a video source and begins playback of a video.

Availability

Flash Lite 2.0

Returns

Boolean - A value of `true` if the mobile device can render the video; otherwise, `false`.

Example

The following example pauses and clears video1.flv, which is playing in a Video object (called `video1`).

```
video1.play( "http://www.macromedia.com/samples/videos/clock.3gp" );
```

You can also use a Video object on the Stage to play bundled device videos directly from the library. To do this, you bundle the device video in your application's library. You also assign an identifier to the video symbol that lets you reference the video symbol with ActionScript. You can play a device video from the library by passing the symbol's ActionScript identifier to the `Video.play()` method, as the following example shows:

```
placeHolderVideo.play("symbol://ocean_video");
```

For more information about playing video from the library, see "Playing a bundled video directly from the library" in *Developing Flash Lite 2.x Applications*.

See also

`stop (Video.stop method)`, `pause (Video.pause method)`, `resume (Video.resume method)`

resume (Video.resume method)

```
public resume() : Void
```

Calling this method resumes playback of the video.

If `Video.pause()` was previously called, playback begins from the current position. If `Video.stop()` was previously called, playback begins from the first frame.

Availability

Flash Lite 2.0

Example

The following example resumes the video that is playing in a Video object called `video1`.

```
video1.resume()
```

See also

`pause (Video.pause method)`, `stop (Video.stop method)`

stop (Video.stop method)

```
public stop() : Void
```

Stops playback of the video and continues to render the current frame onscreen. A subsequent call to `Video.resume()` resumes playback from the first frame of the video.

Availability

Flash Lite 2.0

Example

The following example stops the video that is playing in a Video object (called `my_video`) when the user clicks the `close_btn` instance.

```
// video1 is the name of a Video object on Stage
video1.stop();
```

See also

`play (Video.play method)`, `pause (Video.pause method)`, `resume (Video.resume method)`

XML

```
Object
    |
    +-XMLNode
        |
        +-XML
```

```
public class XML
extends XMLNode
```

Use the methods and properties of the XML class to load, parse, send, build, and manipulate XML document trees.

You must use the constructor `new XML()` to create an XML object before calling any method of the XML class.

An XML document is represented in Flash by the XML class. Each element of the hierarchical document is represented by an XMLNode object.

For information on the following methods and properties, see the XMLNode class: `appendChild()`, `attributes`, `childNodes`, `cloneNode()`, `firstChild`, `hasChildNodes()`, `insertBefore()`, `lastChild`, `nextSibling`, `nodeName`, `nodeType`, `nodeValue`, `parentNode`, `previousSibling`, `removeNode()`, `toString()`

In earlier versions of the ActionScript Language Reference, the methods and properties above were documented in the XML class. They are now documented in the XMLNode class.

Note: The `XML` and XMLNode objects are modeled after the W3C DOM Level 1 recommendation: http://www.w3.org/tr/1998/REC-DOM-Level-1-19981001/level-one-core.html. That recommendation specifies a Node interface and a Document interface. The Document interface inherits from the Node interface, and adds methods such as `createElement()` and `createTextNode()`. In ActionScript, the `XML` and XMLNode objects are designed to divide functionality along similar lines.

Availability

Flash Lite 2.0

See also

`appendChild (XMLNode.appendChild method)`, `attributes (XMLNode.attributes property)`, `childNodes (XMLNode.childNodes property)`, `cloneNode (XMLNode.cloneNode method)`, `firstChild (XMLNode.firstChild property)` `hasChildNodes (XMLNode.hasChildNodes method)`, `insertBefore (XMLNode.insertBefore method)` `lastChild (XMLNode.lastChild property)`, `nextSibling (XMLNode.nextSibling property)` `nodeName (XMLNode.nodeName property)`, `nodeType (XMLNode.nodeType property)` `nodeValue (XMLNode.nodeValue property)`, `parentNode (XMLNode.parentNode property)` `previousSibling (XMLNode.previousSibling property)`, `removeNode (XMLNode.removeNode method)` `toString (XMLNode.toString method)`

Property summary

Modifiers	Property	Description
	contentType : String	The MIME content type that is sent to the server when you call the `XML.send()` or `XML.sendAndLoad()` method.
	docTypeDecl : String	Specifies information about the XML document's `DOCTYPE` declaration.
	ignoreWhite : Boolean	Default setting is `false`.

Modifiers	Property	Description
	loaded : Boolean	Indicates if the XML document has successfully loaded.
	status : Number	Automatically sets and returns a numeric value that indicates whether an XML document was successfully parsed into an XML object.
	xmlDecl : String	A string that specifies information about a document's XML declaration.

Properties inherited from class `XMLNode`

```
attributes (XMLNode.attributes property),childNodes (XMLNode.childNodes
property)firstChild (XMLNode.firstChild property),lastChild (XMLNode.lastChild
property)nextSibling (XMLNode.nextSibling property),nodeName (XMLNode.nodeName
property),nodeType (XMLNode.nodeType property),nodeValue (XMLNode.nodeValue property),
parentNode (XMLNode.parentNode property)previousSibling (XMLNode.previousSibling
property)
```

Properties inherited from class Object

```
constructor (Object.constructor property),__proto__ (Object.__proto__
property)prototype (Object.prototype property),__resolve (Object.__resolve property)
```

Event summary

Event	Description
onData = function(src:String) {}	Invoked when XML text has been completely downloaded from the server, or when an error occurs downloading XML text from a server.
onLoad = function(success:Boolean) {}	Invoked by Flash Lite player when an XML document is received from the server.

Constructor summary

Signature	Description
XML(text:String)	Creates a new XML object.

Method summary

Modifiers	Signature	Description
	addRequestHeader(header:Object, headerValue:String) : Void	Adds or changes HTTP request headers (such as `Content-Type` or `SOAPAction`) sent with `POST` actions.
	createElement(name:String) : XMLNode	Creates a new XML element with the name specified in the parameter.
	createTextNode(value:String) : XMLNode	Creates a new XML text node with the specified text.
	getBytesLoaded() : Number	Returns the number of bytes loaded (streamed) for the XML document.

Modifiers	Signature	Description
	getBytesTotal() : Number	Returns the size, in bytes, of the XML document.
	load(url:String) : Boolean	Loads an XML document from the specified URL, and replaces the contents of the specified XML object with the downloaded XML data.
	parseXML(value:String) : Void	Parses the XML text specified in the value parameter, and populates the specified XML object with the resulting XML tree.
	send(url:String, [target:String], method:String) : Boolean	Encodes the specified XML object into an XML document, and sends it to the specified URL using the POST method in a browser.
	sendAndLoad(url:String, resultXML:XML) : Void	Encodes the specified XML object into an XML document, sends it to the specified URL using the POST method, downloads the server's response, and loads it into resultXMLobject, specified in the parameters.

Methods inherited from class XMLNode

appendChild (XMLNode.appendChild method), cloneNode (XMLNode.cloneNode method) hasChildNodes (XMLNode.hasChildNodes method), insertBefore (XMLNode.insertBefore method) removeNode (XMLNode.removeNode method), toString (XMLNode.toString method)

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

addRequestHeader (XML.addRequestHeader method)

public addRequestHeader(header:Object, headerValue:String) : Void

Adds or changes HTTP request headers (such as Content-Type or SOAPAction) sent with POST actions. In the first usage, you pass two strings (header and headerValue) to the method. In the second usage, you pass an array of strings, alternating header names and header values.

If multiple calls are made to set the same header name, each successive value replaces the value set in the previous call.

You cannot add or change the following standard HTTP headers using this method: Accept-Ranges, Age, Allow, Allowed, Connection, Content-Length, Content-Location, Content-Range, ETag, Host, Last-Modified, Locations, Max-Forwards, Proxy-Authenticate, Proxy-Authorization, Public, Range, Retry-After, Server, TE, Trailer, Transfer-Encoding, Upgrade, URI, Vary, Via, Warning, and WWW-Authenticate.

Availability

Flash Lite 2.0

Parameters

header: Object - A string that represents an HTTP request header name.

headerValue : String - A string that represents the value associated with `header`.

Example

The following example adds a custom HTTP header named `SOAPAction` with a value of `Foo` to an XML object named `my_xml`:

```
my_xml.addRequestHeader("SOAPAction", "'Foo'");
```

The following example creates an array named `headers` that contains two alternating HTTP headers and their associated values. The array is passed as a parameter to the `addRequestHeader()` method.

```
var headers:Array = new Array("Content-Type", "text/plain",
"X-ClientAppVersion", "2.0");
my_xml.addRequestHeader(headers);
```

See also

`addRequestHeader (LoadVars.addRequestHeader method)`

contentType (XML.contentType property)

```
public contentType : String
```

The MIME content type that is sent to the server when you call the `XML.send()` or `XML.sendAndLoad()` method. The default is `application/x-www-form-urlencoded`, which is the standard MIME content type used for most HTML forms.

Availability

Flash Lite 2.0

Example

The following example creates a new XML document and checks its default content type:

```
// create a new XML document
var doc:XML = new XML();

// trace the default content type
trace(doc.contentType); // output: application/x-www-form-urlencoded
```

The following example defines an XML packet, and sets the content type for the XML object. The data is then sent to a server and shows a result in a browser window.

```
var my_xml:XML = new XML("<highscore><name>Ernie</name><score>13045</score>
</highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

Press F12 to test this example in a browser.

See also

`send (XML.send method)`, `sendAndLoad (XML.sendAndLoad method)`

createElement (XML.createElement method)

```
public createElement(name:String) : XMLNode
```

Creates a new XML element with the name specified in the parameter. The new element initially has no parent, no children, and no siblings. The method returns a reference to the newly created XML object that represents the element. This method and the `XML.createTextNode()` method are the constructor methods for creating nodes for an XML object.

Availability
Flash Lite 2.0

Parameters
name : String - The tag name of the XML element being created.

Returns
XMLNode - An XMLNode object; an XML element.

Example
The following example creates three XML nodes using the `createElement()` method:

```
// create an XML document
var doc:XML = new XML();

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

trace(doc);
// output: <element1><element2 /><element3 /></element1>
```

See also
createTextNode (XML.createTextNode method)

createTextNode (XML.createTextNode method)

```
public createTextNode(value:String) : XMLNode
```

Creates a new XML text node with the specified text. The new node initially has no parent, and text nodes cannot have children or siblings. This method returns a reference to the XML object that represents the new text node. This method and the `XML.createElement()` method are the constructor methods for creating nodes for an XML object.

Availability
Flash Lite 2.0

Parameters
value : String - A string; the text used to create the new text node.

Returns

XMLNode - An XMLNode object.

Example

The following example creates two XML text nodes using the `createTextNode()` method, and places them into existing XML nodes:

```
// create an XML document
var doc:XML = new XML();

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

// create two XML text nodes using createTextNode()
var textNode1:XMLNode = doc.createTextNode("textNode1 String value");
var textNode2:XMLNode = doc.createTextNode("textNode2 String value");

// place the new nodes into the XML tree
element2.appendChild(textNode1);
element3.appendChild(textNode2);

trace(doc);
// output (with line breaks added between tags):
// <element1>
// <element2>textNode1 String value</element2>
// <element3>textNode2 String value</element3>
// </element1>
```

See also

createElement (XML.createElement method)

docTypeDecl (XML.docTypeDecl property)

```
public docTypeDecl : String
```

Specifies information about the XML document's `DOCTYPE` declaration. After the XML text has been parsed into an XML object, the `XML.docTypeDecl` property of the XML object is set to the text of the XML document's `DOCTYPE` declaration (for example, `<!DOCTYPEgreeting SYSTEM "hello.dtd">`). This property is set using a string representation of the `DOCTYPE` declaration, not an XML node object.

The ActionScript XML parser is not a validating parser. The `DOCTYPE` declaration is read by the parser and stored in the `XML.docTypeDecl` property, but no DTD validation is performed.

If no `DOCTYPE` declaration was encountered during a parse operation, the `XML.docTypeDecl` property is set to `undefined`. The `XML.toString()` method outputs the contents of `XML.docTypeDecl` immediately after the XML declaration stored in `XML.xmlDecl`, and before any other text in the XML object. If `XML.docTypeDecl` is undefined, no `DOCTYPE` declaration is output.

Availability
Flash Lite 2.0

Example
The following example uses the `XML.docTypeDecl` property to set the `DOCTYPE` declaration for an XML object:

```
my_xml.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

See also
xmlDecl (XML.xmlDecl property)

getBytesLoaded (XML.getBytesLoaded method)

```
public getBytesLoaded() : Number
```

Returns the number of bytes loaded (streamed) for the XML document. You can compare the value of `getBytesLoaded()` with the value of `getBytesTotal()` to determine what percentage of an XML document has loaded.

Availability
Flash Lite 2.0

Returns
Number - An integer that indicates the number of bytes loaded.

Example
The following example shows how to use the `XML.getBytesLoaded()` method with the `XML.getBytesTotal()` method to trace the progress of an `XML.load()` command. You must replace the URL parameter of the `XML.load()` command so that the parameter refers to a valid XML file using HTTP. If you attempt to use this example to load a local file that resides on your hard disk, this example will not work properly because in test movie mode, Flash Lite player loads local files in their entirety.

```
// create a new XML document
var doc:XML = new XML();

var checkProgress = function(xmlObj:XML) {
    var bytesLoaded:Number = xmlObj.getBytesLoaded();
    var bytesTotal:Number = xmlObj.getBytesTotal();
    var percentLoaded:Number = Math.floor((bytesLoaded / bytesTotal ) 100);
    trace ("milliseconds elapsed: " + getTimer());
    trace ("bytesLoaded: " + bytesLoaded);
    trace ("bytesTotal: " + bytesTotal);
    trace ("percent loaded: " + percentLoaded);
    trace ("---------------------------------");
}

doc.onLoad = function(success:Boolean) {
    clearInterval(intervalID);
    trace("intervalID: " + intervalID);
}
doc.load("[place a valid URL pointing to an XML file here]");
var intervalID:Number = setInterval(checkProgress, 100, doc);
```

See also

getBytesTotal (XML.getBytesTotal method)

getBytesTotal (XML.getBytesTotal method)

```
public getBytesTotal() : Number
```

Returns the size, in bytes, of the XML document.

Availability

Flash Lite 2.0

Returns

Number - An integer.

Example

See example for `XML.getBytesLoaded()`.

See also

getBytesLoaded (XML.getBytesLoaded method)

ignoreWhite (XML.ignoreWhite property)

```
public ignoreWhite : Boolean
```

Default setting is `false`. When set to `true`, text nodes that contain only white space are discarded during the parsing process. Text nodes with leading or trailing white space are unaffected.

Usage 1: You can set the `ignoreWhite` property for individual XML objects, as the following code shows:

```
my_xml.ignoreWhite = true;
```

Usage 2: You can set the default `ignoreWhite` property for XML objects, as the following code shows:

```
XML.prototype.ignoreWhite = true;
```

Availability

Flash Lite 2.0

Example

The following example loads an XML file with a text node that contains only white space; the `foyer` tag comprises fourteen space characters. To run this example, create a text file named *flooring.xml*, and copy the following tags into it:

```
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer> </foyer>
</house>
```

Create a new Flash document named *flooring.fla* and save it to the same directory as the XML file. Place the following code into the main Timeline:

```
// create a new XML object
var flooring:XML = new XML();

// set the ignoreWhite property to true (default value is false)
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object
flooring.onLoad = function(success:Boolean) {
    trace(flooring);
}

// load the XML into the flooring object
flooring.load("flooring.xml");

// output (line breaks added for clarity):
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer />
</house>
```

If you then change the setting of `flooring.ignoreWhite` to `false`, or simply remove that line of code entirely, the fourteen space characters in the `foyer` tag will be preserved:

```
...
// set the ignoreWhite property to false (default value)
flooring.ignoreWhite = false;
...
// output (line breaks added for clarity):
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer> </foyer>
</house>
```

For an example, see the XML_blogTracker.fla and XML_languagePicker.fla files in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

load (XML.load method)

```
public load(url:String) : Boolean
```

Loads an XML document from the specified URL, and replaces the contents of the specified XML object with the downloaded XML data. The URL is relative and is called using HTTP. The load process is asynchronous; it does not finish immediately after the `load()` method is executed.

In SWF files running in a version of the player earlier than Flash Player 7, the `url` parameter must be in the same superdomain as the SWF file that issues this call. A *superdomain* is derived by removing the leftmost component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, the `url` parameter must be in exactly the same domain. For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server that is hosting the SWF file.

When the `load()` method is executed, the XML object property `loaded` is set to `false`. When the XML data finishes downloading, the `loaded` property is set to `true`, and the `onLoad` event handler is invoked. The XML data is not parsed until it is completely downloaded. If the XML object previously contained any XML trees, they are discarded.

You can define a custom function that executes when the `onLoad` event handler of the XML object is invoked.

Availability

Flash Lite 2.0

Parameters

url : String - A string that represents the URL where the XML document to be loaded is located. If the SWF file that issues this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see the Description section.

Returns

Boolean - `false` if no parameter (`null`) is passed; `true` otherwise. Use the `onLoad()` event handler to check the success of a loaded XML document.

Example

The following simple example uses the `XML.load()` method:

```
// create a new XML object
var flooring:XML = new XML();

// set the ignoreWhite property to true (default value is false)
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object
flooring.onLoad = function(success) {
    trace(flooring);
};

// load the XML into the flooring object
flooring.load("flooring.xml");
```

For the contents of the flooring.xml file, and the output that this example produces, see the example for `XML.ignoreWhite`.

See also

ignoreWhite (XML.ignoreWhite property), loaded (XML.loaded property), onLoad (XML.onLoad handler)

loaded (XML.loaded property)

```
public loaded : Boolean
```

Indicates if the XML document has successfully loaded. If there is no custom `onLoad()` event handler defined for the XML object, then this property is set to `true` when the document-loading process initiated by the `XML.load()` call has completed successfully; otherwise, it is `false`. However, if you define a custom behavior for the `onLoad()` event handler for the XML object, be sure to set `onload` in that function.

Availability

Flash Lite 2.0

Example

The following example uses the `XML.loaded` property in a simple script:

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
    trace("success: "+success);
    trace("loaded: "+my_xml.loaded);
    trace("status: "+my_xml.status);
};
my_xml.load("http://www.flash-mx.com/mm/problems/products.xml");
```

Information displays in the Output panel when the `onLoad` handler invokes. If the call completes successfully, `true` displays for the `loaded` status in the Output panel.

```
success: true
loaded: true
status: 0
```

See also

load (XML.load method), onLoad (XML.onLoad handler)

onData (XML.onData handler)

```
onData = function(src:String) {}
```

Invoked when XML text has been completely downloaded from the server, or when an error occurs downloading XML text from a server. This handler is invoked before the XML is parsed, and you can use it to call a custom parsing routine instead of using the Flash XML parser. The `src` parameter is a string that contains XML text downloaded from the server, unless an error occurs during the download, in which case the `src` parameter is `undefined`.

By default, the `XML.onData` event handler invokes `XML.onLoad`. You can override the `XML.onData` event handler with custom behavior, but `XML.onLoad` is not called unless you call it in your implementation of `XML.onData`.

Availability

Flash Lite 2.0

Parameters

src : String - A string or `undefined`; the raw data, usually in XML format, that is sent by the server.

Example

The following example shows what the `XML.onData` event handler looks like by default:

```
XML.prototype.onData = function (src:String) {
    if (src == undefined) {
        this.onLoad(false);
    } else {
        this.parseXML(src);
        this.loaded = true;
        this.onLoad(true);
    }
}
```

You can override the `XML.onData` event handler to intercept the XML text without parsing it.

See also

`onLoad (XML.onLoad handler)`

onLoad (XML.onLoad handler)

```
onLoad = function(success:Boolean) {}
```

Invoked by Flash Lite player when an XML document is received from the server. If the XML document is received successfully, the `success` parameter is `true`. If the document was not received, or if an error occurred in receiving the response from the server, the `success` parameter is `false`. The default, implementation of this method is not active. To override the default implementation, you must assign a function that contains custom actions.

Availability

Flash Lite 2.0

Parameters

success : Boolean - A Boolean value that evaluates to `true` if the XML object is successfully loaded with a `XML.load()` or `XML.sendAndLoad()` operation; otherwise, it is `false`.

Example

The following example includes ActionScript for a simple e-commerce storefront application. The `sendAndLoad()` method transmits an XML element that contains the user's name and password, and uses an `XML.onLoad` handler to process the reply from the server.

```
var login_str:String = "<login username=\""+username_txt.text+"\"
password=\""+password_txt.text+"\" />";
var my_xml:XML = new XML(login_str);
var myLoginReply_xml:XML = new XML();

myLoginReply_xml.ignoreWhite = true;

myLoginReply_xml.onLoad = function(success:Boolean){

    if (success) {

        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
            (myLoginReply_xml.firstChild.attributes.success == "true")) {
            gotoAndStop("loggedIn");
        } else {
            gotoAndStop("loginFailed");
        }

    } else {
        gotoAndStop("connectionFailed");
    }

};

my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm", myLoginReply_xml);
```

See also

load (XML.load method), sendAndLoad (XML.sendAndLoad method)

parseXML (XML.parseXML method)

```
public parseXML(value:String) : Void
```

Parses the XML text specified in the `value` parameter, and populates the specified XML object with the resulting XML tree. Any existing trees in the XML object are discarded.

Availability

Flash Lite 2.0

Parameters

value : String - A string that represents the XML text to be parsed and passed to the specified XML object.

Example

The following example creates and parses an XML packet:

```
var xml_str:String = "<state name=\"California\">
<city>San Francisco</city></state>"

// defining the XML source within the XML constructor:
var my1_xml:XML = new XML(xml_str);
trace(my1_xml.firstChild.attributes.name); // output: California

// defining the XML source using the XML.parseXML method:
var my2_xml:XML = new XML();
my2_xml.parseXML(xml_str);
trace(my2_xml.firstChild.attributes.name); // output: California
```

send (XML.send method)

```
public send(url:String, [target:String], method:String) : Boolean
```

Encodes the specified XML object into an XML document, and sends it to the specified URL using the `POST` method in a browser. The Flash test environment only uses the `GET` method.

Availability

Flash Lite 2.0

Parameters

url : String - String; the destination URL for the specified XML object.

target : String [optional] - String; the browser window to show data that the server returns:

- `_self` specifies the current frame in the current window.
- `_blank` specifies a new window.
- `_parent` specifies the parent of the current frame.
- `_top` specifies the top-level frame in the current window.

  If you do not specify a `window` parameter, it is the same as specifying `_self`.

method : String [optional] - the method of the HTTP protocol used: either `GET` or `POST`. In a browser, the default value is `POST`. In the Flash test environment, the default value is `GET`.

Returns

Boolean - `false` if no parameters are specified, `true` otherwise

Example

The following example defines an XML packet and sets the content type for the XML object. The data is then sent to a server and shows a result in a browser window.

```
var my_xml:XML = new XML("<highscore><name>Ernie</name>
<score>13045</score></highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

Press F12 to test this example in a browser.

See also

sendAndLoad (XML.sendAndLoad method)

sendAndLoad (XML.sendAndLoad method)

```
public sendAndLoad(url:String, resultXML:XML) : Void
```

Encodes the specified XML object into an XML document, sends it to the specified URL using the `POST` method, downloads the server's response, and loads it into the `resultXMLobject` specified in the parameters. The server response loads in the same manner used by the `XML.load()` method.

In SWF files running in a version of the player earlier than Flash Player 7, the `url` parameter must be in the same superdomain as the SWF file that is issuing this call. A *superdomain* is derived by removing the leftmost component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, the `url` parameter must be in exactly the same domain. For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file.

When `sendAndLoad()` is executed, the XML object property `loaded` is set to `false`. When the XML data finishes downloading, the `loaded` property is set to `true` if the data successfully loaded, and the `onLoad` event handler is invoked. The XML data is not parsed until it is completely downloaded. If the XML object previously contained any XML trees, they are discarded.

Availability

Flash Lite 2.0

Parameters

url : String - A string; the destination URL for the specified XML object. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see the Description section.

resultXML : XML - A target XML object created with the XML constructor method that will receive the return information from the server.

Example

The following example includes ActionScript for a simple e-commerce storefront application. The `XML.sendAndLoad()` method transmits an XML element that contains the user's name and password, and uses an `onLoad` handler to process the reply from the server.

```
var login_str:String = "<login username=\""+username_txt.text+"\"
password=\""+password_txt.text+"\" />";
var my_xml:XML = new XML(login_str);
var myLoginReply_xml:XML = new XML();
myLoginReply_xml.ignoreWhite = true;
myLoginReply_xml.onLoad = myOnLoad;
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm", myLoginReply_xml);
function myOnLoad(success:Boolean) {
    if (success) {
    if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
        (myLoginReply_xml.firstChild.attributes.success == "true")) {
        gotoAndStop("loggedIn");
    } else {
        gotoAndStop("loginFailed");
    }
    } else {
    gotoAndStop("connectionFailed");
    }
}
```

See also

send (XML.send method), load (XML.load method), loaded (XML.loaded property), onLoad (XML.onLoad handler)

status (XML.status property)

```
public status : Number
```

Automatically sets and returns a numeric value that indicates whether an XML document was successfully parsed into an XML object. The following are the numeric status codes, with descriptions:

- 0 No error; parse was completed successfully.
- -2 A CDATA (character data) section was not properly terminated.
- -3 The `XML` declaration was not properly terminated.
- -4 The `DOCTYPE` declaration was not properly terminated.
- -5 A comment was not properly terminated.
- -6 An XML element was malformed.
- -7 Out of memory.
- -8 An attribute value was not properly terminated.
- -9 A start-tag was not matched with an end-tag.
- -10 An end-tag was encountered without a matching start-tag.

Availability

Flash Lite 2.0

Example

The following example loads an XML packet into a SWF file. A status message displays, depending on whether the XML loads and parses successfully. Add the following ActionScript to your FLA or AS file:

```
var my_xml:XML = new XML();
my_xml.onLoad = function(success:Boolean) {
    if (success) {
    if (my_xml.status == 0) {
        trace("XML was loaded and parsed successfully");
    } else {
        trace("XML was loaded successfully, but was unable to be parsed.");
    }
    var errorMessage:String;
    switch (my_xml.status) {
    case 0 :
        errorMessage = "No error; parse was completed successfully.";
        break;
    case -2 :
        errorMessage = "A CDATA section was not properly terminated.";
        break;
    case -3 :
        errorMessage = "The XML declaration was not properly terminated.";
        break;
    case -4 :
        errorMessage = "The DOCTYPE declaration was not properly terminated.";
        break;
    case -5 :
        errorMessage = "A comment was not properly terminated.";
        break;
    case -6 :
        errorMessage = "An XML element was malformed.";
        break;
    case -7 :
        errorMessage = "Out of memory.";
        break;
    case -8 :
        errorMessage = "An attribute value was not properly terminated.";
        break;
    case -9 :
        errorMessage = "A start-tag was not matched with an end-tag.";
        break;
    case -10 :
        errorMessage = "An end-tag was encountered without a matching
        start-tag.";
        break;
    default :
        errorMessage = "An unknown error has occurred.";
        break;
    }
    trace("status: "+my_xml.status+" ("+errorMessage+")");
    } else {
    trace("Unable to load/parse XML. (status: "+my_xml.status+")");
    }
};
my_xml.load("http://www.helpexamples.com/flash/badxml.xml");
```

XML constructor

```
public XML(text:String)
```

Creates a new XML object. You must use the constructor to create an XML object before you call any of the methods of the XML class.

Note: Use the `createElement()` and `createTextNode()` methods to add elements and text nodes to an XML document tree.

Availability
Flash Lite 2.0

Parameters
text : String - A string; the XML text parsed to create the new XML object.

Example
The following example creates a new, empty XML object:

```
var my_xml:XML = new XML();
```

The following example creates an XML object by parsing the XML text specified in the `source` parameter, and populates the newly created XML object with the resulting XML document tree:

```
var other_xml:XML = new XML("<state name=\"California\"><city>San Francisco</city></state>");
```

See also
`createElement (XML.createElement method)`, `createTextNode (XML.createTextNode method)`

xmlDecl (XML.xmlDecl property)

```
public xmlDecl : String
```

A string that specifies information about a document's XML declaration. After the XML document is parsed into an XML object, this property is set to the text of the document's XML declaration. This property is set using a string representation of the XML declaration, not an XML node object. If no XML declaration is encountered during a parse operation, the property is set to `undefined`.`XML`. The `XML.toString()` method outputs the contents of the `XML.xmlDecl` property before any other text in the XML object. If the `XML.xmlDecl` property contains the `undefined` type, no XML declaration is output.

Availability
Flash Lite 2.0

Example
The following example creates a text field called `my_txt` that has the same dimensions as the Stage. The text field displays properties of the XML packet that loads into the SWF file. The doc type declaration displays in `my_txt`. Add the following ActionScript to your FLA or AS file:

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "_typewriter";
my_fmt.size = 12;
my_fmt.leftMargin = 10;

this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, Stage.width, Stage.height);
my_txt.border = true;
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.setNewTextFormat(my_fmt);

var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
    var endTime:Number = getTimer();
    var elapsedTime:Number = endTime-startTime;
    if (success) {
    my_txt.text = "xmlDecl:"+newline+my_xml.xmlDecl+newline+newline;
    my_txt.text += "contentType:"+newline+my_xml.contentType+newline+newline;
    my_txt.text += "docTypeDecl:"+newline+my_xml.docTypeDecl+newline+newline;
    my_txt.text += "packet:"+newline+my_xml.toString()+newline+newline;
    } else {
    my_txt.text = "Unable to load remote XML."+newline+newline;
    }
    my_txt.text += "loaded in: "+elapsedTime+" ms.";
};
my_xml.load("http://www.helpexamples.com/crossdomain.xml");
var startTime:Number = getTimer();
```

See also

docTypeDecl (XML.docTypeDecl property)

XMLNode

```
Object
    |
    +-XMLNode

public class XMLNode
extends Object
```

An XML document is represented in Flash by the XML class. Each element of the hierarchical document is represented by an XMLNode object.

Availability

Flash Lite 2.0

See also

hasXMLSocket (capabilities.hasXMLSocket property)

Property summary

Modifiers	Property	Description
	attributes : Object	An object containing all of the attributes of the specified XML instance.
	childNodes : Array [read-only]	An array of the specified XML object's children.
	firstChild : XMLNode [read-only]	Evaluates the specified XML object and references the first child in the parent node's child list.
	lastChild : XMLNode [read-only]	An XMLNode value that references the last child in the node's child list.
	nextSibling : XMLNode [read-only]	An XMLNode value that references the next sibling in the parent node's child list.
	nodeName : String	A string representing the node name of the XML object.
	nodeType : Number [read-only]	A `nodeType` value, either 1 for an XML element or 3 for a text node.
	nodeValue : String	The node value of the XML object.
	parentNode : XMLNode [read-only]	An XMLNode value that references the parent node of the specified XML object, or returns `null` if the node has no parent.
	previousSibling : XMLNode [read-only]	An XMLNode value that references the previous sibling in the parent node's child list.

Properties inherited from class Object

```
constructor (Object.constructor property),__proto__ (Object.__proto__
property)prototype (Object.prototype property),__resolve (Object.__resolve property)
```

Method summary

Modifiers	Signature	Description
	appendChild(newChild: XMLNode) : Void	Appends the specified node to the XML object's child list.
	cloneNode(deep:Boolean) : XMLNode	Constructs and returns a new XML node of the same type, name, value, and attributes as the specified XML object.
	hasChildNodes() : Boolean	Specifies whether or not the XML object has child nodes.
	insertBefore(newChild: XMLNode, insertPoint:XMLNode) : Void	Inserts a `newChild` node into the XML object's child list, before the `insertPoint` node.
	removeNode() : Void	Removes the specified XML object from its parent.
	toString() : String	Evaluates the specified XML object, constructs a textual representation of the XML structure, including the node, children, and attributes, and returns the result as a string.

Methods inherited from class Object

```
addProperty (Object.addProperty method),hasOwnProperty (Object.hasOwnProperty
method)isPropertyEnumerable (Object.isPropertyEnumerable method)isPrototypeOf
(Object.isPrototypeOf method)registerClass (Object.registerClass method),toString
(Object.toString method)unwatch (Object.unwatch method),valueOf (Object.valueOf
method)watch (Object.watch method)
```

appendChild (XMLNode.appendChild method)

```
public appendChild(newChild:XMLNode) : Void
```

Appends the specified node to the XML object's child list. This method operates directly on the node referenced by the `childNode` parameter; it does not append a copy of the node. If the node to be appended already exists in another tree structure, appending the node to the new location will remove it from its current location. If the `childNode` parameter refers to a node that already exists in another XML tree structure, the appended child node is placed in the new tree structure after it is removed from its existing parent node.

Availability

Flash Lite 2.0

Parameters

newChild : XMLNode - An XMLNode object that represents the node to be moved from its current location to the child list of the my_xml object.

Example

This example does the following things in the order shown:

- Creates two empty XML documents, `doc1` and `doc2`.
- Creates a new node using the `createElement()` method and appends it, using the `appendChild()` method, to the XML document named `doc1`.
- Shows how to move a node using the `appendChild()` method by moving the root node from `doc1` to `doc2`.
- Clones the root node from `doc2` and appends it to `doc1`.
- Creates a new node and appends it to the root node of the XML document `doc1`.

```
var doc1:XML = new XML();
var doc2:XML = new XML();

// create a root node and add it to doc1
var rootnode:XMLNode = doc1.createElement("root");
doc1.appendChild(rootnode);
trace ("doc1: " + doc1); // output: doc1: <root />
trace ("doc2: " + doc2); // output: doc2:

// move the root node to doc2
doc2.appendChild(rootnode);
trace ("doc1: " + doc1); // output: doc1:
trace ("doc2: " + doc2); // output: doc2: <root />

// clone the root node and append it to doc1
var clone:XMLNode = doc2.firstChild.cloneNode(true);
doc1.appendChild(clone);
trace ("doc1: " + doc1); // output: doc1: <root />
trace ("doc2: " + doc2); // output: doc2: <root />

// create a new node to append to root node (named clone) of doc1
var newNode:XMLNode = doc1.createElement("newbie");
clone.appendChild(newNode);
trace ("doc1: " + doc1); // output: doc1: <root><newbie /></root>
```

attributes (XMLNode.attributes property)

```
public attributes : Object
```

An object containing all of the attributes of the specified XML instance. The `XML.attributes` object contains one variable for each attribute of the XML instance. Because these variables are defined as part of the object, they are generally referred to as properties of the object. The value of each attribute is stored in the corresponding property as a string. For example, if you have an attribute named `color`, you would retrieve that attribute's value by specifying color as the property name, as the following code shows:

```
var myColor:String = doc.firstChild.attributes.color
```

Availability

Flash Lite 2.0

Example

The following example shows the XML attribute names:

```
// create a tag called 'mytag' with
// an attribute called 'name' with value 'Val'
var doc:XML = new XML("<mytag name=\"Val\"> item </mytag>");

// assign the value of the 'name' attribute to variable y
var y:String = doc.firstChild.attributes.name;
trace (y); // output: Val

// create a new attribute named 'order' with value 'first'
doc.firstChild.attributes.order = "first";

// assign the value of the 'order' attribute to variable z
var z:String = doc.firstChild.attributes.order
trace(z); // output: first
```

The following is displayed in the Output panel:

```
Val
first
```

childNodes (XMLNode.childNodes property)

```
public childNodes : Array [read-only]
```

An array of the specified XML object's children. Each element in the array is a reference to an XML object that represents a child node. This is a read-only property and cannot be used to manipulate child nodes. Use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

This property is undefined for text nodes (`nodeType == 3`).

Availability

Flash Lite 2.0

Example

The following example shows how to use the `XML.childNodes` property to return an array of child nodes:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// create an array and use rootNode to populate it
var firstArray:Array = doc.childNodes;
trace (firstArray);
// output: <rootNode><oldest /><middle /><youngest /></rootNode>

// create another array and use the child nodes to populate it
var secondArray:Array = rootNode.childNodes;
trace(secondArray);
// output: <oldest />,<middle />,<youngest />
```

See also

nodeType (XMLNode.nodeType property), appendChild (XMLNode.appendChild method), insertBefore (XMLNode.insertBefore method) removeNode (XMLNode.removeNode method)

cloneNode (XMLNode.cloneNode method)

```
public cloneNode(deep:Boolean) : XMLNode
```

Constructs and returns a new XML node of the same type, name, value, and attributes as the specified XML object. If `deep` is set to `true`, all child nodes are recursively cloned, resulting in an exact copy of the original object's document tree.

The clone of the node that is returned is no longer associated with the tree of the cloned item. Consequently, `nextSibling`, `parentNode`, and `previousSibling` all have a value of `null`. If the `deep` parameter is set to `false`, or the `my_xml` node has no child nodes, `firstChild` and `lastChild` are also null.

Availability

Flash Lite 2.0

Parameters

deep : Boolean - A Boolean value; if set to `true`, the children of the specified XML object will be recursively cloned.

Returns

XMLNode - An XMLNode object.

Example

The following example shows how to use the `XML.cloneNode()` method to create a copy of a node:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// create a copy of the middle node using cloneNode()
var middle2:XMLNode = middle.cloneNode(false);

// insert the clone node into rootNode between the middle and youngest nodes
rootNode.insertBefore(middle2, youngest);
trace(rootNode);
// output (with line breaks added):
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// </rootNode>

// create a copy of rootNode using cloneNode() to demonstrate a deep copy
var rootClone:XMLNode = rootNode.cloneNode(true);

// insert the clone, which contains all child nodes, to rootNode
rootNode.appendChild(rootClone);
trace(rootNode);
// output (with line breaks added):
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// </rootNode>
// </rootNode>
```

firstChild (XMLNode.firstChild property)

```
public firstChild : XMLNode [read-only]
```

Evaluates the specified XML object and references the first child in the parent node's child list. This property is `null` if the node does not have children. This property is `undefined` if the node is a text node. This is a read-only property and cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Availability

Flash Lite 2.0

Example

The following example shows how to use `XML.firstChild` to loop through a node's child nodes:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use firstChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode = aNode.nextSibling) {
    trace(aNode);
}

// output:
// <oldest />
// <middle />
// <youngest />
```

The following example is from the XML_languagePicker.fla file in the Examples directory and can be found in the `languageXML.onLoad` event handler function definition:

```
// loop through the strings in each language node
// adding each string as a new element in the language array
for (var stringNode:XMLNode = childNode.firstChild; stringNode != null; stringNode =
stringNode.nextSibling, j++) {
    masterArray[i][j] = stringNode.firstChild.nodeValue;
}
```

To view the entire script, see XML_languagePicker.fla in the ActionScript samples folder at www.adobe.com/go/learn_fl_samples. Download and decompress the .zip file and navigate to the folder for your version of ActionScript to access the sample.

See also

appendChild (XMLNode.appendChild method), insertBefore (XMLNode.insertBefore method), removeNode (XMLNode.removeNode method)

hasChildNodes (XMLNode.hasChildNodes method)

```
public hasChildNodes() : Boolean
```

Specifies whether or not the XML object has child nodes.

Availability

Flash Lite 2.0

Returns

Boolean - `true` if the specified XMLNode object has one or more child nodes; otherwise `false`.

Example

The following example creates a new XML packet. If the root node has child nodes, the code loops over each child node to display the name and value of the node. Add the following ActionScript to your FLA or AS file:

```
var my_xml:XML = new XML("hankrudolph");
if (my_xml.firstChild.hasChildNodes()) {
// use firstChild to iterate through the child nodes of rootNode
    for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
aNode=aNode.nextSibling) {
        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

The following is displayed in the Output panel:

```
output:
username: hank
password: rudolph
```

insertBefore (XMLNode.insertBefore method)

```
public insertBefore(newChild:XMLNode, insertPoint:XMLNode) : Void
```

Inserts a `newChild` node into the XML object's child list, before the `insertPoint` node. If `insertPoint` is *not* a child of the XMLNode object, the insertion fails.

Availability

Flash Lite 2.0

Parameters

newChild : XMLNode - The XMLNode object to be inserted.

insertPoint : XMLNode - The XMLNode object that will follow the `newChild` node after the method is invoked.

Example

The following inserts a new XML node between two existing nodes:

```
var my_xml:XML = new XML("<a>1</a>\n<c>3</c>");
var insertPoint:XMLNode = my_xml.lastChild;
var newNode:XML = new XML("<b>2</b>\n");
my_xml.insertBefore(newNode, insertPoint);
trace(my_xml);
```

See also

hasXMLSocket (capabilities.hasXMLSocket property), cloneNode (XMLNode.cloneNode method)

lastChild (XMLNode.lastChild property)

```
public lastChild : XMLNode [read-only]
```

An XMLNode value that references the last child in the node's child list. The `XML.lastChild` property is `null` if the node does not have children. This property cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Availability

Flash Lite 2.0

Example

The following example uses the `XML.lastChild` property to iterate through the child nodes of an XML node, beginning with the last item in the node's child list and ending with the first child of the node's child list:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode = aNode.previousSibling) {
    trace(aNode);
}

// output:
// <youngest />
// <middle />
// <oldest />
```

The following example creates a new XML packet and uses the `XML.lastChild` property to iterate through the child nodes of the root node:

```
// create a new XML document
var doc:XML = new XML("");

var rootNode:XMLNode = doc.firstChild;

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode=aNode.previousSibling) {
    trace(aNode);
}

// output:
// <youngest />
// <middle />
// <oldest />
```

See also

appendChild (XMLNode.appendChild method), insertBefore (XMLNode.insertBefore method), removeNode (XMLNode.removeNode method), hasXMLSocket (capabilities.hasXMLSocket property)

nextSibling (XMLNode.nextSibling property)

```
public nextSibling : XMLNode [read-only]
```

An XMLNode value that references the next sibling in the parent node's child list. This property is `null` if the node does not have a next sibling node. This property cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Availability

Flash Lite 2.0

Example

The following example is an excerpt from the example for the `XML.firstChild` property, and shows how you can use the `XML.nextSibling` property to loop through an XML node's child nodes:

```
for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode = aNode.nextSibling) {
    trace(aNode);
}
```

See also

firstChild (XMLNode.firstChild property), appendChild (XMLNode.appendChild method), insertBefore (XMLNode.insertBefore method), removeNode (XMLNode.removeNode method), hasXMLSocket (capabilities.hasXMLSocket property)

nodeName (XMLNode.nodeName property)

```
public nodeName : String
```

A string representing the node name of the XML object. If the XML object is an XML element (`nodeType == 1`), `nodeName` is the name of the tag that represents the node in the XML file. For example, `TITLE` is the `nodeName` of an HTML `TITLE` tag. If the XML object is a text node (`nodeType == 3`), `nodeName` is `null`.

Availability

Flash Lite 2.0

Example

The following example creates an element node and a text node, and checks the node name of each:

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeName);
trace(myTextNode.nodeName);

// output:
// rootNode
// null
```

The following example creates a new XML packet. If the root node has child nodes, the code loops over each child node to display the name and value of the node. Add the following ActionScript to your FLA or AS file:

```
var my_xml:XML = new XML("hankrudolph");
if (my_xml.firstChild.hasChildNodes()) {
    // use firstChild to iterate through the child nodes of rootNode
    for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
aNode=aNode.nextSibling) {
        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

The following node names are displayed in the Output panel:

```
output:
username: hank
password: rudolph
```

See also

nodeType (XMLNode.nodeType property)

nodeType (XMLNode.nodeType property)

```
public nodeType : Number [read-only]
```

A `nodeType` value, either 1 for an XML element or 3 for a text node.

The `nodeType` is a numeric value from the NodeType enumeration in the W3C DOM Level 1 recommendation at www.w3.org. The following table lists the values:

Integer value	Defined constant
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	NTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INStrUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

In Flash Lite player, the built-in XML class only supports 1 (`ELEMENT_NODE`) and 3 (`TEXT_NODE`).

Availability

Flash Lite 2.0

Example

The following example creates an element node and a text node, and checks the node type of each:

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeType);
trace(myTextNode.nodeType);

// output:
// 1
// 3
```

See also

nodeValue (XMLNode.nodeValue property)

nodeValue (XMLNode.nodeValue property)

```
public nodeValue : String
```

The node value of the XML object. If the XML object is a text node, the `nodeType` is 3, and the `nodeValue` is the text of the node. If the XML object is an XML element (`nodeType` is 1), `nodeValue` is `null` and read-only

Availability

Flash Lite 2.0

Example

The following example creates an element node and a text node, and checks the node value of each:

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeValue);
trace(myTextNode.nodeValue);

// output:
// null
// myTextNode
```

The following example creates and parses an XML packet. The code loops through each child node, and displays the node value using the `firstChild` property and `firstChild.nodeValue`. When you use `firstChild` to display contents of the node, it maintains the `&` entity. However, when you explicitly use `nodeValue`, it converts to the ampersand character (&).

```
var my_xml:XML = new XML("mortongood&evil");
trace("using firstChild:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
    trace("\t"+my_xml.firstChild.childNodes[i].firstChild);
}
trace("");
trace("using firstChild.nodeValue:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
    trace("\t"+my_xml.firstChild.childNodes[i].firstChild.nodeValue);
}
```

The following information is displayed in the Output panel:

```
using firstChild:
    morton
    good&evil

using firstChild.nodeValue:
    morton
    good&evil
```

See also

nodeType (XMLNode.nodeType property)

parentNode (XMLNode.parentNode property)

```
public parentNode : XMLNode [read-only]
```

An XMLNode value that references the parent node of the specified XML object, or returns `null` if the node has no parent. This is a read-only property and cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Availability

Flash Lite 2.0

Example

The following example creates an XML packet and displays the parent node of the username node in the Output panel:

```
var my_xml:XML = new XML("mortongood&evil");

// first child is the <login /> node
var rootNode:XMLNode = my_xml.firstChild;

// first child of the root is the <username /> node
var targetNode:XMLNode = rootNode.firstChild;
trace("the parent node of '"+targetNode.nodeName+"' is: "+targetNode.parentNode.nodeName);
trace("contents of the parent node are:\n"+targetNode.parentNode);

// output (line breaks added for clarity):

the parent node of 'username' is: login
contents of the parent node are:
    morton
    good&evil
```

See also

appendChild (XMLNode.appendChild method), insertBefore (XMLNode.insertBefore method), removeNode (XMLNode.removeNode method), hasXMLSocket (capabilities.hasXMLSocket property)

previousSibling (XMLNode.previousSibling property)

```
public previousSibling : XMLNode [read-only]
```

An XMLNode value that references the previous sibling in the parent node's child list. The property has a value of `null` if the node does not have a previous sibling node. This property cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Availability
Flash Lite 2.0

Example
The following example is an excerpt from the example for the `XML.lastChild` property, and shows how you can use the `XML.previousSibling` property to loop through an XML node's child nodes:

```
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode = aNode.previousSibling) {
    trace(aNode);
}
```

See also
lastChild (XMLNode.lastChild property), appendChild (XMLNode.appendChild method), insertBefore (XMLNode.insertBefore method), removeNode (XMLNode.removeNode method), hasXMLSocket (capabilities.hasXMLSocket property)

removeNode (XMLNode.removeNode method)

```
public removeNode() : Void
```

Removes the specified XML object from its parent. Also deletes all descendants of the node.

Availability
Flash Lite 2.0

Example
The following example creates an XML packet, and then deletes the specified XML object and its descendant nodes:

```
var xml_str:String = "<state name=\"California\"><city>San Francisco</city></state>";

var my_xml:XML = new XML(xml_str);
var cityNode:XMLNode = my_xml.firstChild.firstChild;
trace("before XML.removeNode():\n"+my_xml);
cityNode.removeNode();
trace("");
trace("after XML.removeNode():\n"+my_xml);

// output (line breaks added for clarity):
//
// before XML.removeNode():
// <state name="California">
// <city>San Francisco</city>
// </state>
//
// after XML.removeNode():
// <state name="California" />
```

toString (XMLNode.toString method)

```
public toString() : String
```

Evaluates the specified XML object, constructs a textual representation of the XML structure, including the node, children, and attributes, and returns the result as a string.

For top-level XML objects (those created with the constructor), the `XML.toString()` method outputs the document's XML declaration (stored in the `XML.xmlDecl` property), followed by the document's `DOCTYPE` declaration (stored in the `XML.docTypeDecl` property), followed by the text representation of all XML nodes in the object. The XML declaration is not output if the `XML.xmlDecl` property is undefined. The `DOCTYPE` declaration is not output if the `XML.docTypeDecl` property is `undefined`.

Availability
Flash Lite 2.0

Returns
String - String.

Example
The following code uses the `toString()` method to convert an XMLNode object to a String, and then uses the `toUpperCase()` method of the String class:

```
var xString = "<first>Mary</first>"
        + "<last>Ng</last>"
var my_xml:XML = new XML(xString);
var my_node:XMLNode = my_xml.childNodes[1];
trace(my_node.toString().toUpperCase());
        // <LAST>NG<
```

See also
docTypeDecl (XML.docTypeDecl property), xmlDecl (XML.xmlDecl property)

XMLSocket

```
Object
    |
    +-XMLSocket

public class XMLSocket
extends Object
```

The XMLSocket class implements client sockets that let the device running the Flash Lite player communicate with a server computer that an IP address or domain name identifies. The XMLSocket class is useful for client-server applications that require low latency, such as real-time chat systems. A traditional HTTP-based chat system frequently polls the server and downloads new messages by using an HTTP request. In contrast, an XMLSocket chat solution maintains an open connection to the server, which lets the server immediately send incoming messages without a request from the client.

To use the XMLSocket class, the server computer must run a daemon process that understands the protocol that the XMLSocket class uses. The following list describes the protocol:

- XML messages are sent over a full-duplex TCP/IP stream socket connection.
- Each XML message is a complete XML document, terminated by a zero (0) byte.
- An unlimited number of XML messages can be sent and received over a single XMLSocket connection.

The following restrictions apply to how and where an XMLSocket object can connect to the server:

- To connect an XMLSocket to a port lower than 1024, you must first load a policy file with the `System.security.loadPolicyFile()` method, even when your application connects to its own exact domain.
- The `XMLSocket.connect()` method can connect only to computers in the same domain where the SWF file resides. This restriction does not apply to SWF files running on a local disk. (This restriction is identical to the security rules for the `loadVariables()` function, and the `XML.sendAndLoad()` and `XML.load()` methods.) To connect to a server daemon running in a domain other than the one where the SWF file resides, you can create a security policy file on the server that allows access from specific domains.

Setting up a server to communicate with the XMLSocket object can be challenging. If your application does not require real-time interactivity, use the `loadVariables()` function, or Flash HTTP-based XML server connectivity methods (`XML.load()`, `XML.sendAndLoad()`, `XML.send()`), instead of the XMLSocket class.

To use the methods of the XMLSocket class, you must first use the constructor, `XMLSocket()`, to create an XMLSocket object.

Availability

Flash Lite 2.1

See also

`loadPolicyFile (security.loadPolicyFile method)`

Property summary

Properties inherited from class Object

`constructor (Object.constructor property), __proto__ (Object.__proto__ property) prototype (Object.prototype property), __resolve (Object.__resolve property)`

Event summary

Event	Description
onClose = function() {}	Invoked only when the server closes an open connection.
onConnect = function(success:Boolean) {}	An asynchronous callback that the Flash Lite player invokes when a connection request initiated through `XMLSocket.connect()` succeeds or fails.
onData = function(src:String) {}	Invoked when a message is downloaded from the server and terminated by a zero (0) byte.
onXML = function(src:XML) {}	Invoked by the Flash Lite player when the specified XML object containing an XML document arrives over an open XMLSocket connection.

Constructor summary

Signature	Description
XMLSocket()	Creates a new XMLSocket object.

Method summary

Modifiers	Signature	Description
	close() : Void	Closes the connection that the XMLSocket object specifies.
	connect(url:String, port:Number) : Boolean	Establishes a connection to the specified Internet host by using the specified TCP port and returns true or false, depending on whether a connection is successfully initiated.
	send(data:Object) : Void	Converts the XML object or data specified in the object parameter to a string and transmits it to the server, followed by a zero (0) byte.

Methods inherited from class Object

addProperty (Object.addProperty method), hasOwnProperty (Object.hasOwnProperty method) isPropertyEnumerable (Object.isPropertyEnumerable method) isPrototypeOf (Object.isPrototypeOf method) registerClass (Object.registerClass method), toString (Object.toString method) unwatch (Object.unwatch method), valueOf (Object.valueOf method) watch (Object.watch method)

close (XMLSocket.close method)

```
public close() : Void
```

Closes the connection that the XMLSocket object specifies.

Availability

Flash Lite 2.1

Example

The following simple example creates an XMLSocket object, attempts to connect to the server, and then closes the connection.

```
var socket:XMLSocket = new XMLSocket();
socket.connect(null, 2000);
socket.close();
```

See also

connect (XMLSocket.connect method)

connect (XMLSocket.connect method)

```
public connect(url:String, port:Number) : Boolean
```

Establishes a connection to the specified Internet host by using the specified TCP port and returns `true` or `false`, depending on whether a connection is successfully initiated. If the `XMLSocket.connect()` method returns a value of `true`, the initial stage of the connection process is successful; later, the `XMLSocket.onConnect()` method is invoked to determine whether the final connection succeeded or failed. If `XMLSocket.connect()` returns `false`, a connection could not be established.

If you do not know the port number of your Internet host computer, contact your network administrator. To connect an XMLSocket to a port lower than 1024, you must first load a policy file with the `System.security.loadPolicyFile()` method.

If you specify `null` for the `host` parameter, the host contacted is the one where the SWF file calling `XMLSocket.connect()` resides. For example, if the SWF file was downloaded from www.example.com, specifying `null` for the host parameter is the same as entering the IP address for www.example.com.

In SWF files of any version running in Flash Player 7 or later, the `host` parameter must be in exactly the same domain. For example, a SWF file at www.someDomain.com that is published for Flash Player 5, but is running in Flash Player 7 or later, can load variables only from SWF files that are also at www.someDomain.com. If you want to load variables from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file that is being accessed.

Note: The `XMLSocket.connect()` method returns `false` if `System.capabilities.hasXMLSocket` is `false`.

Availability
Flash Lite 2.1

Parameters
url : String - String; A fully qualified DNS domain name or an IP address in the form *aaa.bbb.ccc.ddd*. You can also specify `null` to connect to the host server on which the SWF file resides. If the SWF file issuing this call is running in a web browser, the `host` parameter must be in the same domain as the SWF file.

port : Number - Number; The TCP port number on the host used to establish a connection.

Returns
Boolean - A value of `true` if the connection is successful; `false` otherwise.

Example
The following example uses the `XMLSocket.connect()` method to connect to the host where the SWF file resides and uses the `trace()` function to display the return value indicating the success or failure of the connection:

```
var socket:XMLSocket = new XMLSocket()
socket.onConnect = function (success:Boolean) {
    if (success) {
    trace ("Connection succeeded!");
    } else {
    trace ("Connection failed!");
    }
}
if (!socket.connect(null, 2000)) {
    trace ("Connection failed!");
}
```

See also
onConnect (XMLSocket.onConnect handler), Array function, loadPolicyFile (security.loadPolicyFile method)

onClose (XMLSocket.onClose handler)

```
onClose = function() {}
```

Invoked only when the server closes an open connection. The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing custom actions.

Availability

Flash Lite 2.1

Example

The following example executes a trace statement if the server closes an open connection:

```
var socket:XMLSocket = new XMLSocket();
socket.connect(null, 2000);
socket.onClose = function () {
    trace("Connection to server lost.");
}
```

See also

`onConnect (XMLSocket.onConnect handler)`, `Array function`

onConnect (XMLSocket.onConnect handler)

```
onConnect = function(success:Boolean) {}
```

An asynchronous callback that the Flash Lite player invokes when a connection request initiated through `XMLSocket.connect()` succeeds or fails. If the connection succeeds, the `success` parameter has a value of `true`; otherwise the `success` parameter has a value of `false`.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing custom actions.

Availability

Flash Lite 2.1

Parameters

success : Boolean - A Boolean value indicating whether a socket connection is successfully established. If the connection succeeded, the `success` parameter has a value of `true`; otherwise the `success` parameter has a value of `false`.

Example

The following example illustrates the process of specifying a replacement function for the `onConnect()` event handler in a simple chat application.

After creating the XMLSocket object by using the constructor method, the script defines the custom function to be executed when the `onConnect()` event handler is invoked. The function controls the screen to which users are taken, depending on whether a connection is successfully established. If the connection is successfully made, users are taken to the main chat screen on the frame labeled startChat. If the connection is not successful, users go to a screen with troubleshooting information on the frame labeled connectionFailed.

```
var socket:XMLSocket = new XMLSocket();
socket.onConnect = function (success) {
    if (success) {
        gotoAndPlay("startChat");
    } else {
        gotoAndStop("connectionFailed");
    }
}
```

Now that the `onConnect()` handler is defined, the connect() method is invoked to attempt to establish the connection. If the `connect()` method returns a value of `false`, the SWF file is sent directly to the frame labeled connectionFailed, and `onConnect()` is never invoked. If the `connect()` method returns `true`, the SWF file jumps to a frame labeled waitForConnection, which is the "Please wait" screen. The SWF file remains on the waitForConnection frame until the `onConnect()` handler is invoked, which happens at some point in the future depending on network latency.

```
if (!socket.connect(null, 2000)) {
    gotoAndStop("connectionFailed");
} else {
    gotoAndStop("waitForConnection");
}
```

See also

`connect (XMLSocket.connect method)`, `Array function`

onData (XMLSocket.onData handler)

```
onData = function(src:String) {}
```

Invoked when a message is downloaded from the server and terminated by a zero (0) byte. You can override the `XMLSocket.onData` event handler to intercept data that the server sends without parsing it as XML. This capability is useful if you're transmitting arbitrarily formatted data packets, and you'd prefer to manipulate the data directly when it arrives, rather than have Flash Lite player parse the data as XML.

By default, the `XMLSocket.onData` method invokes the `XMLSocket.onXML` method. If you override `XMLSocket.onData` with custom behavior, `XMLSocket.onXML` is not called unless you call it in your implementation of `XMLSocket.onData`.

Availability

Flash Lite 2.1

Parameters

src : String - A string containing data that the server sends.

Example

In this example, the `src` parameter is a string containing XML text downloaded from the server. The zero (0) byte terminator is not included in the string.

```
XMLSocket.prototype.onData = function (src) {
    this.onXML(new XML(src));
}
```

onXML (XMLSocket.onXML handler)

```
onXML = function(src:XML) {}
```

Invoked by the Flash Lite player when the specified XML object containing an XML document arrives over an open XMLSocket connection. An XMLSocket connection can be used to transfer an unlimited number of XML documents between the client and the server. Each document is terminated with a zero (0) byte. When the Flash Lite player receives the zero byte, it parses all the XML received since the previous zero byte or since the connection was established if this is the first message received. Each batch of parsed XML is treated as a single XML document and passed to the `onXML()` method.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing actions that you define.

Availability
Flash Lite 2.1

Parameters
src : XML - An XML object that contains a parsed XML document received from a server.

Example
The following function overrides the default implementation of the `onXML()` method in a simple chat application. The `myOnXML()` function instructs the chat application to recognize a single XML element, `MESSAGE`, in the following format:

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />
var socket:XMLSocket = new XMLSocket();
```

In the following example, the `displayMessage()` function is assumed to be a user-defined function that displays the message that the user receives:

```
socket.onXML = function (doc) {
    var e = doc.firstChild;
    if (e != null && e.nodeName == "MESSAGE") {
        displayMessage(e.attributes.user, e.attributes.text);
    }
}
```

See also
`Array function`

send (XMLSocket.send method)

```
public send(data:Object) : Void
```

Converts the XML object or data specified in the `object` parameter to a string and transmits it to the server, followed by a zero (0) byte. If `object` is an XML object, the string is the textual representation of the XML object. The send operation is asynchronous; it returns immediately, but the data can be transmitted at a later time. The `XMLSocket.send()` method does not return a value indicating whether the data was successfully transmitted.

If the XMLSocket object is not connected to the server (using the `XMLSocket.connect()` method), the `XMLSocket.send()` operation fails.

Availability
Flash Lite 2.1

Parameters
data : Object - An XML object or other data to transmit to the server.

Example
The following example shows how to specify a user name and password to send the `my_xml` XML object to the server:

```
var myXMLSocket:XMLSocket = new XMLSocket();
var my_xml:XML = new XML();
var myLogin:XMLNode = my_xml.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
my_xml.appendChild(myLogin);
myXMLSocket.send(my_xml);
```

See also

connect (XMLSocket.connect method)

XMLSocket constructor

```
public XMLSocket()
```

Creates a new XMLSocket object. The XMLSocket object is not initially connected to any server. You must call the `XMLSocket.connect()` method to connect the object to a server.

Availability

Flash Lite 2.1

Example

The following example creates an XMLSocket object:

```
var socket:XMLSocket = new XMLSocket();
```

Chapter 3: Deprecated ActionScript

The evolution of ActionScript has deprecated many elements of the language. This section lists the deprecated items and suggests alternatives when available. While deprecated elements still work in Flash Lite 2.0 and later, Adobe recommends that you do not continue using deprecated elements in your code. Support of deprecated elements in the future is not guaranteed.

Deprecated Function summary

Modifiers	Function Name	Description
	call(frame:Object)	Deprecated since Flash Player 5. This action was deprecated in favor of the `function` statement.
	chr(number:Number)String	Deprecated since Flash Player 5. This function was deprecated in favor of `String.fromCharCode()`.
	getProperty(my_mc:Object, property:Object)Object	Deprecated since Flash Player 5. This function was deprecated in favor of the dot syntax, which was introduced in Flash Player 5.
	ifFrameLoaded([scene:String], frame:Object, statement(s):Object)	Deprecated since Flash Player 5. Adobe recommends that you use the `MovieClip._framesloaded` property.
	int(value:Number)Number	Deprecated since Flash Player 5. This function was deprecated in favor of `Math.round()`.
	length(expression:String, variable:Object)Number	Deprecated since Flash Player 5. This function, along with all the string functions, has been deprecated. Adobe recommends that you use the methods of the String class and the `String.length` property to perform the same operations.
	mbchr(number:Number)	Deprecated since Flash Player 5. This function was deprecated in favor of the `String.fromCharCode()` method.
	mblength(string:String)Number	Deprecated since Flash Player 5. This function was deprecated in favor of the `String.length` property.
	mbord(character:String)Number	Deprecated since Flash Player 5. This function was deprecated in favor of `String.charCodeAt()` method.
	mbsubstring(value:String, index:Number, count:Number)String	Deprecated since Flash Player 5. This function was deprecated in favor of `String.substr()` method.
	ord(character:String)Number	Deprecated since Flash Player 5. This function was deprecated in favor of the methods and properties of the String class.
	random(value:Number)Number	Deprecated since Flash Player 5. This function was deprecated in favor of `Math.random()`.

Modifiers	Function Name	Description
	substring(string:String, index:Number, count:Number)String	Deprecated since Flash Player 5. This function was deprecated in favor of `String.substr()`.
	tellTarget(target:String, statement(s):Object)	Deprecated since Flash Player 5. Adobe recommends that you use dot (.) notation and the `with` statement.
	toggleHighQuality()	Deprecated since Flash Player 5. This function was deprecated in favor of `_quality`.

Deprecated Property summary

Modifiers	Property Name	Description
	$version	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.version` property.
	_cap4WayKeyAS	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.has4WayKeyAS` property.
	_capCompoundSound	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasCompoundSound` property.
	_capEmail	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasEmail` property.
	_capLoadData	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasDataLoading` property.
	_capMFi	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMFi` property.
	_capMIDI	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMIDI` property.
	_capMMS	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasMMS` property.
	_capSMAF	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasSMAF` property.
	_capSMS	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasSMS` property.
	_capStreamSound	Deprecated since Flash Lite Player 2.0. This action was deprecated in favor of the `System.capabilities.hasStreamingAudio` property.
	Button._highquality	Deprecated since Flash Player 7. This property was deprecated in favor of `Button._quality`.

Modifiers	Property Name	Description
	MovieClip._highquality	Deprecated since Flash Player 7. This property was deprecated in favor of `MovieClip._quality`.
	TextField._highquality	Deprecated since Flash Player 7. This property was deprecated in favor of `TextField._quality`.
	_highquality	Deprecated since Flash Player 5. This property was deprecated in favor of `_quality`.
	maxscroll	Deprecated since Flash Player 5. This property was deprecated in favor of `TextField.maxscroll`.
	scroll	Deprecated since Flash Player 5. This property was deprecated in favor of `TextField.scroll`.

Deprecated Operator summary

Operator	Description
<> (inequality)	Deprecated since Flash Player 5. Adobe recommends that you use the `!=` (inequality) operator.
add (concatenation (strings))	Deprecated since Flash Player 5. Adobe recommends you use the addition (+) operator when creating content for Flash Player 5 or later. **Note**: Flash Lite 2.0 also deprecates the `add` operator in favor of the addition (+) operator.
and (logical AND)	Deprecated since Flash Player 5. Adobe recommends that you use the logical AND (`&&`) operator.
eq (equality (strings))	Deprecated since Flash Player 5. This operator was deprecated in favor of the == (equality) operator.
ge (greater than or equal to (strings))	Deprecated since Flash Player 5. This operator was deprecated in favor of the >= (greater than or equal to) operator.
gt (greater than (strings))	Deprecated since Flash Player 5. This operator was deprecated in favor of the > (greater than) operator.
le (less than or equal to (strings))	Deprecated since Flash Player 5. This operator was deprecated in Flash 5 in favor of the <= (less than or equal to) operator.
lt (less than (strings))	Deprecated since Flash Player 5. This operator was deprecated in favor of the < (less than) operator.
ne (not equal (strings))	Deprecated since Flash Player 5. This operator was deprecated in favor of the `!=` (inequality) operator.
not (logical NOT)	Deprecated since Flash Player 5. This operator was deprecated in favor of the `!` (logical NOT) operator.
or (logical OR)	Deprecated since Flash Player 5. This operator was deprecated in favor of the || (logical OR) operator.

Chapter 4: Unsupported ActionScript

The following lists show the classes, methods, properties, global functions, event handlers, and fscommands that are supported in ActionScript 2.0 but are not supported in any version of Flash Lite. For a more detailed presentation of this information, see Supported, partially supported, and unsupported ActionScript classes and language elements in Developing Adobe Flash Lite 2.x and 3.x Applications.

Unsupported Classes

Accessibility, BevelFilter, BitmapFilter, BlurFilter, Camera, ColorMatrixFilter, ContextMenu, ContextMenuItem, ConvolutionFilter, CustomActions, DisplacementMapFilter, DropShadowFilter, FileReference, FileReferenceList, GlowFilter, GradientBevelFilter, GradientGlowFilter, IME, Locale (mx.lang.Locale), Microphone, PrintJob, TextField.StyleSheet, TextRenderer, TextSnapshot, XMLUI

Unsupported Methods

BitmapData.applyFilter, BitmapData.generateFilterRect, BitmapData.noise, BitmapData.paletteMap, BitmapData.perlinNoise, BitmapData.pixelDissolve, BitmapData.scroll, BitmapData.threshold, Matrix.createGradientBox, Mouse.hide, Mouse.show, MovieClip.attachAudio, MovieClip.getTextSnapshot, Selection.getBeginIndex, Selection.getCaretIndex, Selection.getEndIndex, System.setClipboard, System.showSettings, TextField.getFontList, Video.clear

Unsupported Properties

Button.blendMode, Button.cacheAsBitmap, Button.filters, Button.menu, Button.useHandCursor, System.capabilities.language, System.capabilities.manufacturer, System.capabilities.pixelAspectRatio, System.capabilities.playerType, System.capabilities.screenColor, System.capabilities.screenDPI, System.capabilities.serverString, Key.isToggled, MovieClip.menu, MovieClip.useHandCursor, Stage.showMenu, System.exactSettings, TextField.menu, TextField.mouseWheelEnabled, TextField.restrict, Video._alpha, Video.deblocking, Video._height, Video.height, Video._name, Video._parent, Video._rotation, Video.smoothing, Video._visible, Video._width, Video.width, Video._x, Video._xmouse, Video._xscale, Video._y, Video._ymouse, Video._yscale

Unsupported Global Functions

asfunction, MMExecute, print, printAsBitmap, printAsBitmapNum, printNum, updateAfterEvent

Unsupported Event Handlers

onUpdate, Mouse.onMouseWheel

Unsupported fscommands

allowscale, exec, fullscreen, quit, showmenu, trapallkeys

Index

A

a property 366
add() method 510
allowDomain event 337
allowInsecureDomain event 340
alphaMultiplier property 265
alphaOffset property 266
attachBitmap() method 394

B

b property 366
BitmapData
  BitmapData() constructor 200
  clone() method 201
  colorTransform() method 203
  copyChannel() method 203
  copyPixels() method 204
  dispose() method 205
  draw() method 206
  fillRect() method 207
  floodFill() method 208
  getColorBoundsRect() method 209
  getPixel() method 210
  getPixel32() method 211
  height property 212
  hitTest() method 212
  loadBitmap() method 214
  merge() method 215
  rectangle property 216
  setPixel() method 217
  setPixel32() method 217
  transparent property 217
BitmapData() constructor 200
blueMultiplier property 266
blueOffset property 267
bottom property 519
bottomRight property 519

C

c property 367
class
  dispatched by LocalConnection 337, 340, 345
clone() method 201, 367, 510, 520
close() method 341
ColorTransform
  alphaMultiplier property 265
  alphaOffset property 266
  blueMultiplier property 266
  blueOffset property 267
  ColorTransform() constructor 268
  concat() method 269
  greenMultiplier property 270
  greenOffset property 271
  redMultiplier property 272
  redOffset property 273
  rgb property 274
  toString() method 275
ColorTransform class 263
colorTransform property 653
ColorTransform() constructor 268
colorTransform() method 203
concat() method 269, 368
concatenatedColorTransform property 654
concatenatedMatrix property 655
connect() method 341
contains() method 522
containsPoint() method 523
containsRectangle() method 524
ContextMenu class 275
copyChannel() method 203
copyPixels() method 204
createBox() method 369
createGradientBox() method 370

D

d property 371
deltaTransformPoint() method 371
dispose() method 205
distance() method 511
domain() method 342
draw() method 206

E

equals() method 511, 524

F

fillRect() method 207
floodFill() method 208

G

getColorBoundsRect() method 209
getPixel() method 210
getPixel32() method 211
greenMultiplier property 270
greenOffset property 271

H

height property 212, 525
hitTest() method 212

I

identity() method 372
inflate() method 526
inflatePoint() method 527
interpolate() method 512
intersection() method 527
intersects() method 528
invert() method 374
isEmpty() method 529

L

left property 529
length property 512
loadBitmap() method 214
LocalConnection
  close() method 341
  connect() method 341
  domain() method 342
  LocalConnection() constructor 344
  send() method 346
LocalConnection class 336
  allowDomain event 337
  allowInsecureDomain event 340
  onStatus event 345
LocalConnection() constructor 344

M

Matrix
  a property 366
  b property 366
  c property 367
  clone() method 367
  concat() method 368
  createBox() method 369

createGradientBox() method 370
d property 371
deltaTransformPoint() method 371
identity() method 372
invert() method 374
Matrix() constructor 375
rotate() method 376
scale() method 378
toString() method 379
transformPoint() method 379
translate() method 380
tx property 381
ty property 382
Matrix class 362
matrix property 656
Matrix() constructor 375
merge() method 215
MovieClip
attachBitmap() method 394
transform property 452

N

NetConnection
NetConnection() constructor 473
NetConnection() constructor 473
NetStream
NetStream() constructor 479
NetStream() constructor 479
normalize() method 513

O

offset() method 513, 530
offsetPoint() method 530
onStatus event 345

P

pixelBounds property 657
Point
add() method 510
clone() method 510
distance() method 511
equals() method 511
interpolate() method 512
length property 512
normalize() method 513
offset() method 513
Point() constructor 514
polar() method 514
subtract() method 515
toString() method 516
x property 516
y property 516
Point() constructor 514
polar() method 514
PrintJob class 516

R

Rectangle
bottom property 519
bottomRight property 519
clone() method 520
contains() method 522
containsPoint() method 523
containsRectangle() method 524
equals() method 524
height property 525
inflate() method 526
inflatePoint() method 527
intersection() method 527
intersects() method 528
isEmpty() method 529
left property 529
offset() method 530
offsetPoint() method 530
Rectangle() constructor 531
right property 532
setEmpty() method 532
size property 533
top property 533
topLeft property 534
toString() method 535
union() method 535
width property 536
x property 537
y property 537
rectangle property 216
Rectangle() constructor 531
redMultiplier property 272
redOffset property 273
rgb property 274
right property 532
rotate() method 376

S

scale() method 378
send() method 346
setEmpty() method 532
setPixel() method 217
setPixel32() method 217
size property 533
subtract() method 515

T

top property 533
topLeft property 534
toString() method 275, 379, 516, 535
Transform
colorTransform property 653
concatenatedColorTransform property 654
concatenatedMatrix property 655
matrix property 656
pixelBounds property 657
Transform() constructor 658
Transform class 652
transform property 452
Transform() constructor 658
transformPoint() method 379
translate() method 380
transparent property 217
tx property 381
ty property 382

U

union() method 535

W

width property 536

X

x property 516, 537

Y

y property 516, 537